THE MICROSOFT JOURNAL FOR DEVELOPERS

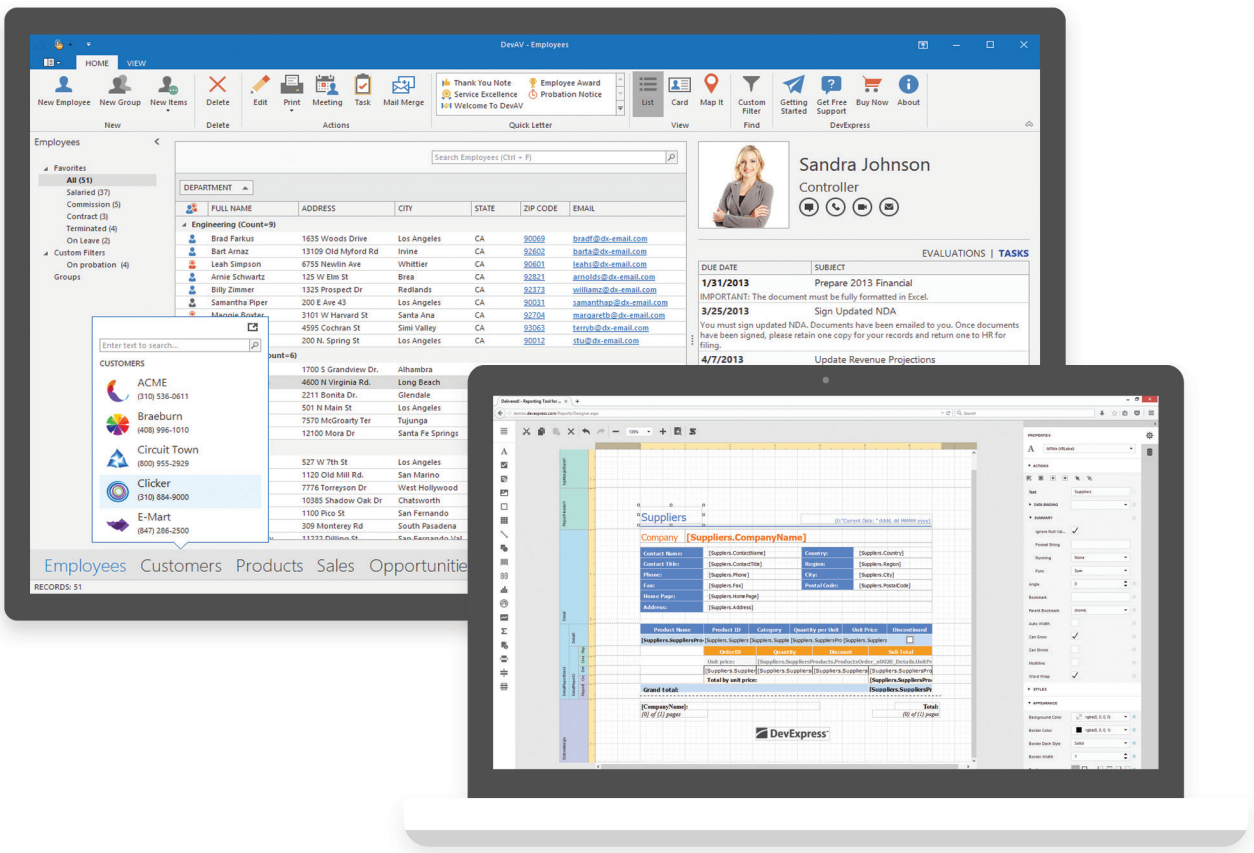
DECEMBER 2018 VOL 33 NO 12

msdn magazine

DevExpress®

Your Next Great App Starts Here

From apps that replicate the look and feel of Microsoft Office® to high-powered data mining and decision support systems for your enterprise, DevExpress UI components for desktops, the web and mobile world will help you build your best, without limits or compromise.



Experience the DevExpress Difference
Download Your Free 30-Day Trial Today

devexpress.com/trial

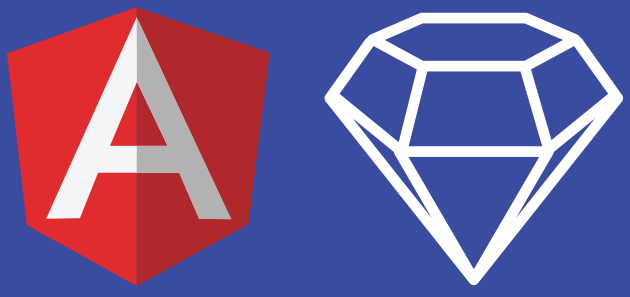
THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 2018 VOL 33 NO 12

msdn magazine

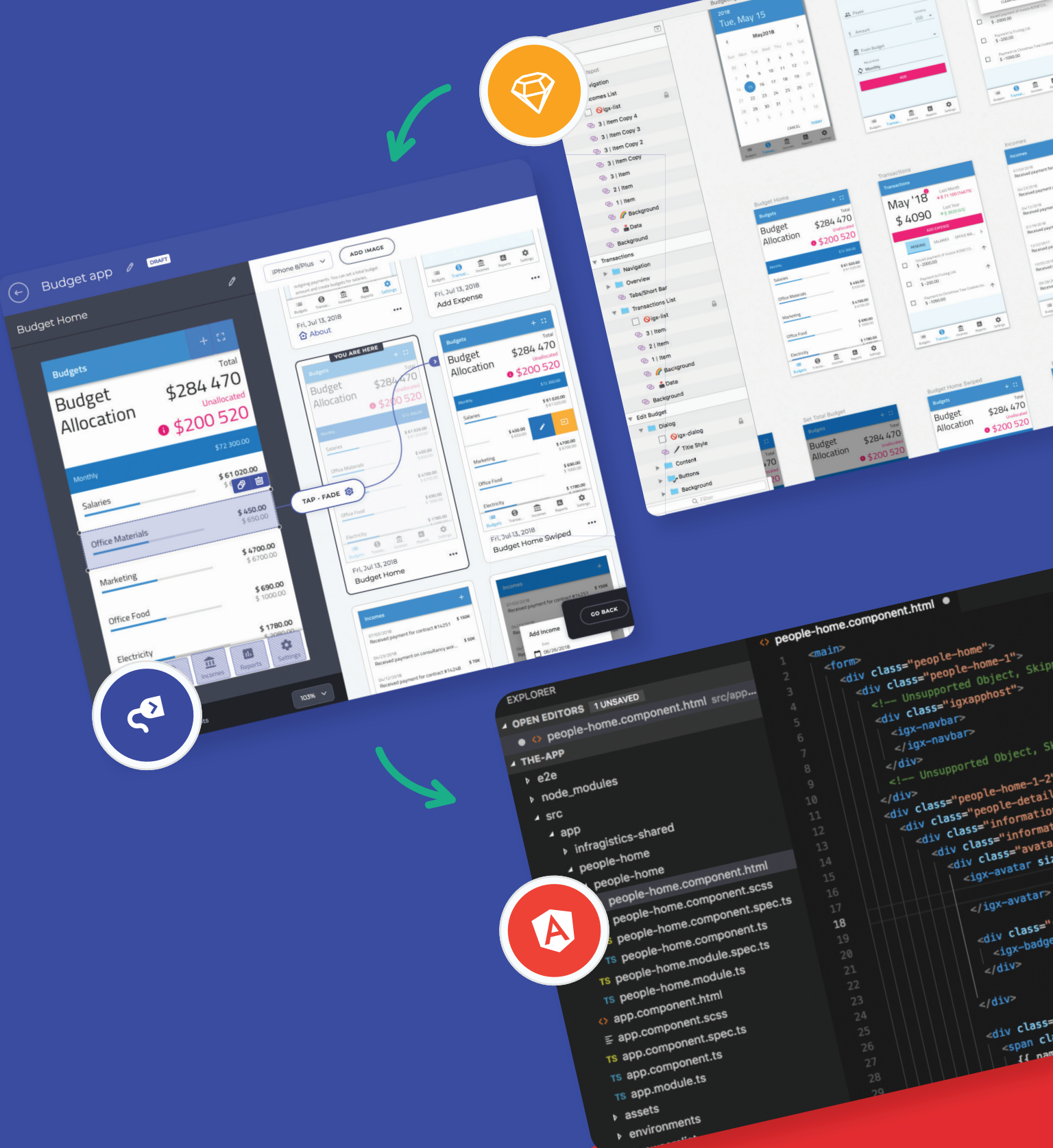
COLUMNS

Microsoft



Get Angular Code From Sketch Designs

With **Usability Studies** and **Collaboration** in the Cloud, plus the fastest Data Grid and Charts for any modern web & mobile experience!



indigo.design

Get it today for only **$99/month**
visit **indigo.design**

msdn magazine

DECEMBER 2018 VOLUME 33 NUMBER 12

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 6300 Canoga Avenue, Suite 1150, Woodland Hills, CA 91367. Periodicals postage paid at Woodland Hills, CA 91367 and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. $35.00, International $60.00. Annual digital subscription rates payable in U.S. funds are: U.S. $25.00, International $25.00. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call 866-293-3194 or 847-513-6011 for U.S. & Canada; 00-1-847-513-6011 for International, fax 847-763-9564. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

CORPORATE ADDRESS 1105 Media, Inc.
6300 Canoga Avenue, Suite 1150, Woodland Hills 91367
www.1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@Converge360.com

REPRINTS For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International
Phone: 212-221-9595
E-mail: 1105reprints@parsintl.com
www.1105Reprints.com

LIST RENTAL This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Jane Long, Merit Direct.
Phone: (913) 685-1301;
Email: jlong@meritdirect.com;
Web: www.meritdirect.com/1105

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail.

E-mail: To e-mail any member of the staff, please use the following form: FirstinitialLastname@1105media.com

Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
2121 Alton Pkwy., Suite240, Irvine, CA 92606

Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
6300 Canoga Ave., Suite 1150, Woodland Hills, CA 91367

The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

LEADTOOLS®

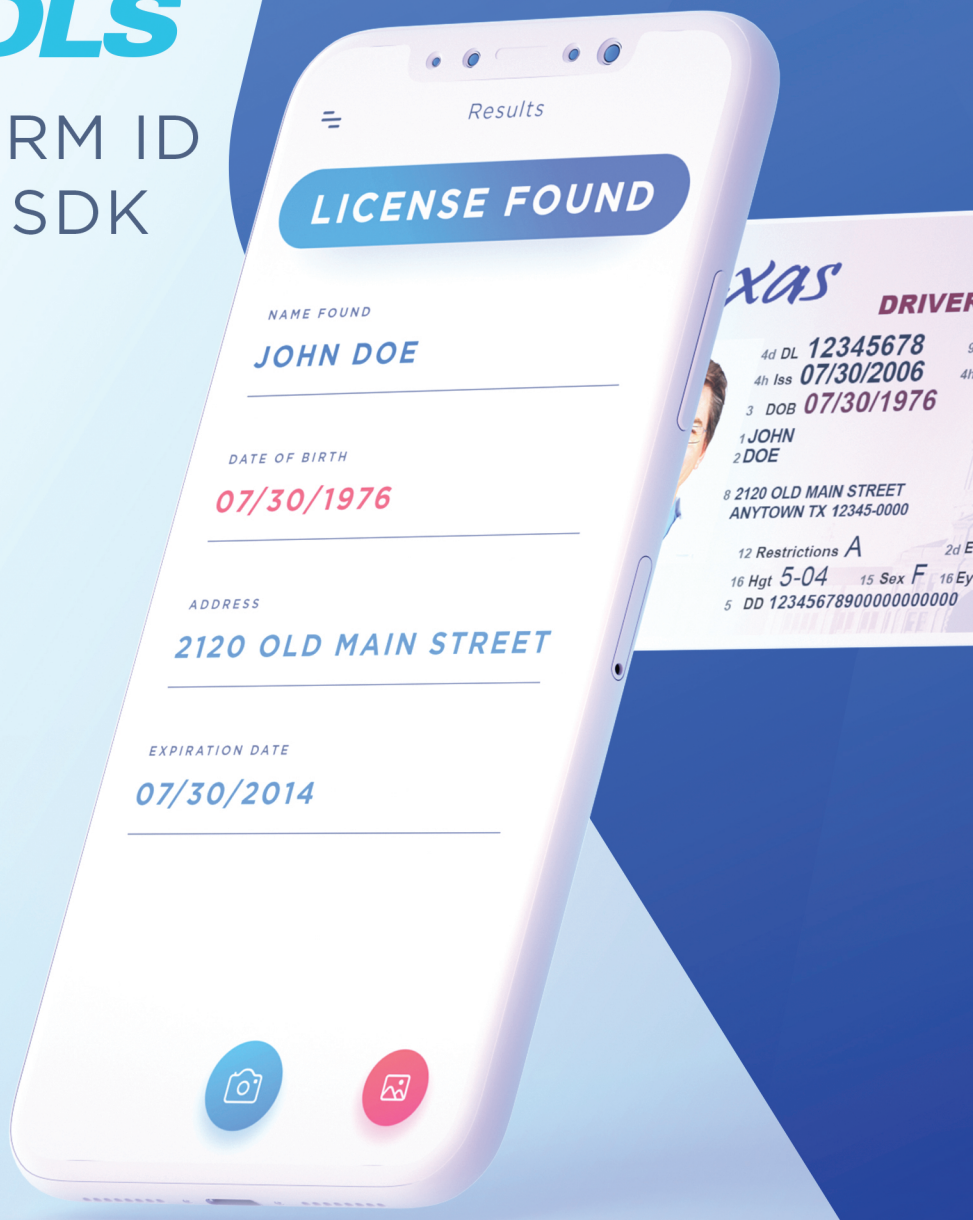
MULTI-PLATFORM ID RECOGNITION SDK

WEB

MOBILE

DESKTOP

SERVER



LEADTOOLS ID recognition and processing SDK technology provides a high-level framework to classify and extract data from driver's licenses and similar forms of identification. The state-of-the-art algorithms recognize structured and unstructured passports, driver's licenses, and ID cards from any country, state, or agency.

Microsoft .NET · X · Windows · Java · Linux · iOS · macOS · Android · HTML5

Get Started Today

DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM

MICHAEL DESMOND

End of an Era

It's not unusual for *MSDN Magazine* column authors to embark on in-depth explorations that span multiple issues. These series can sometimes carry on for three, four or more installments. Heck, Dino Esposito's Cutting Edge column is right now in the throes of a multi-part spelunking of the .NET-savvy Blazor framework for Web app development.

Even by those standards, what Ted Neward has done with his extended series on the MEAN stack (MongoDB, Express, Angular, Node.js) is extravagant. He kicked off the first in his "How To Be MEAN" series of columns way back in August 2015 (msdn.com/magazine/mt185576). Now, 30 columns later, it all comes to a finish with the aptly named "How To Be MEAN: End-to-End."

I asked Neward to single out his favorite columns in the series, but he declined, saying, "Honestly, that's sort of hard. Kind of like with kids, you love them all, even if some are more difficult to work with than others." He admits, though, to a server-side bias, which made getting up to speed with Angular's front-end antics both challenging and ultimately rewarding.

"I'm not as 'hip' to some of the browser-side things people are doing these days, like virtual DOM and such. One of the nice parts of learning about Angular, though, was the realization that it can abstract most (if not all) of that away, and leave me working exclusively in a land of components. That was a nice realization."

The evolving nature of the stack could be an obstacle. AngularJS introduced a component-based application architecture with version 1.6, but Angular 2 in September 2016 was a sharp break that put some adopters in a bind. Neward also points out that when he launched the series, full-stack frameworks like MEAN.io were commanding attention in the Node.js space. Now, he says, those solutions have "almost completely faded from view," as developers worry less about hiding the network between single-page application frameworks and the server.

During the course of the MEAN series, Neward says that in some ways the MEAN stack has simplified, with Angular taking on some of the steps needed to build out the stack. But it has grown more complex in others. As Neward puts it:

> AngularJS introduced a component-based application architecture with version 1.6, but Angular 2 in September 2016 was a sharp break that put some adopters in a bind.

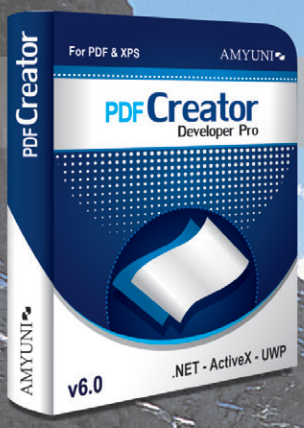
"MEAN now doesn't always mean Mongo-Express-Angular-Node, but rather 'some kind of single-page applications framework and some kind of Node.js back end and some kind of NoSQL storage,'" Neward says. "When a term loses some of its precision, we lose the ability to communicate about it (and around it) effectively."

So MEAN is finished. What's next for the column? Neward urges readers to e-mail him with topics at ted@tedneward.com. In the meantime he has a few ideas: "A working programmer's work is never done. I'll talk a bit about WebAssembly, maybe explore some CI/CD pipeline tools, as well as the odd programming language or two."

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

Switch to Amyuni® PDF

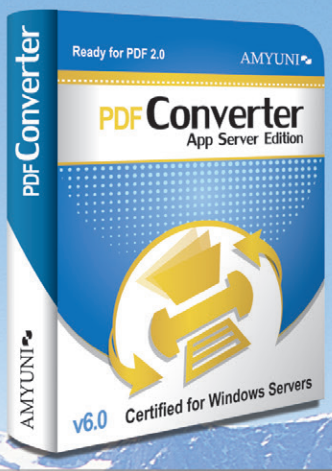
AMYUNI



Create and Process PDFs in .NET, COM/ActiveX and UWP

NEW v6.0

- Edit, process and print PDF 2.0 documents
- Create, fill-out and annotate PDF forms with Javascript support
- Fast and lightweight 64-bit components
- Universal Apps DLLs enable publishing C#, C++, CX or JS apps to Windows Store
- Plus a number of exciting features in v6.0



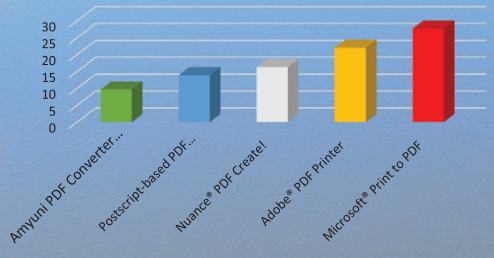
High Performance PDF Printer for Desktops and Servers

Print to PDF in a fraction of the time needed with other tools. WHQL certified for all Windows platforms. Version 6.0 updated for Server 2016.

Complete Suite of PDF, XPS and DOCX Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment
- Includes our Microsoft® WHQL certified PDF and DOCX printer driver
- Export PDF documents into other formats such as images, HTML5 or editable DOCX
- Import/Export XPS files using our native libraries

Benchmark Testing - Amyuni vs Others
Seconds required to convert a document to PDF



Other Developer Components from Amyuni®

- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract 3 engine for accurate character recognition
- Javascript Engine: Integrate a full JS interpreter into your PDF processing applications
- USB Mobile Monitor: Mirror the display of your Windows or Linux system onto your Android device

Windows Server 2016 Certified

Windows Server 2012 Certified

Windows

All development tools available at
www.amyuni.com

AMYUNI Technologies

USA and Canada
Toll Free: 1866 926 9864
Support: 514 868 9227
sales@amyuni.com

Europe
UK: 0800-015-4682
Germany: 0800-183-0923
France: 0800-911-248

JULIE LERMAN

Manage Data Across Multiple Sources with Azure Data Studio

Just about a year ago, I introduced you to the new SQL Operations Studio, a cross-platform, lightweight IDE for working with various flavors of SQL Server (bit.ly/2RlZKuW). SQL Operations Studio was in preview at the time, and has since gone through many transformations. It was officially released at the Microsoft Ignite conference in September, renamed as Azure Data Studio. While the new name doesn't mean the elimination of non-Azure SQL Server usage, I'm personally hoping it does suggest that, someday, we'll be able to use the product for other types of data stores on Azure. Compared to the version I wrote about a year ago, the biggest story regarding the new release is the extensions that are now available.

Rather than repeat information about features that existed in the earlier version, in this article I'll provide a light overview for those of you who are totally new to the application and then explore some of the extensions that piqued my developer brain.

Azure Data Studio is billed as a complementary app to SQL Server Management Studio (SSMS), with a focus on being a low-impact way to execute queries. I think there will be a subset of folks who use both, with Azure Data Studio additionally finding a user base of developers who've never used SSMS. While there are definitely features that will be of interest to DBAs, SSMS will always be the deep-dive administration tool.

Azure Data Studio grew from the intersection of the cross-platform Visual Studio Code (VS Code) and its mssql extension—which I wrote about in my June 2017 column (msdn.com/magazine/mt809115)—eventually becoming its own application. Also, like VS Code, Azure Data Studio is open source and you can get involved at github.com/microsoft/azuredatastudio.

You can download the relevant installers for Windows, macOS or Linux from bit.ly/2Rl14yl. To give you an idea of how different the installation experience is from SSMS, the installer file for the Windows app is 76MB, compared to 800MB for SSMS 17.9, and it installed on my laptop in less than two minutes. Like VS Code, Azure Data Studio is easily configurable through its JSON configuration files, as well as being quite extensible. Azure Data Studio already has more than a dozen extensions, some from Microsoft and some from the community.

Another important feature inherited from VS Code is the interactive terminal where you can run commands from CLIs like PowerShell, Bash and the cross-platform sqlcmd command-line utility. Azure Data Studio also inherits the VS Code File Explorer, which lets you interact with folder-based projects that contain files such as SQL you've written or text-based data files. And you can use the built-in source control to manage and share those projects.

I won't go over the Azure Data Studio capabilities I wrote about in the earlier SQL Operations Studio or mssql articles. There, you can read about such features as the built-in SQL snippets, the ability to create your own custom snippets, and the amazing graphing feature that lets you visualize query results and even embed them as custom widgets on the dashboards for a server or a database.

The experience of using Azure Data Studio is the same on any supported OS. I'll use my MacBook because it's a fun change of venue for me. As I walk through some of the features, I'll assume you have some familiarity with getting around VS Code or earlier versions of SQL Operations Studio.

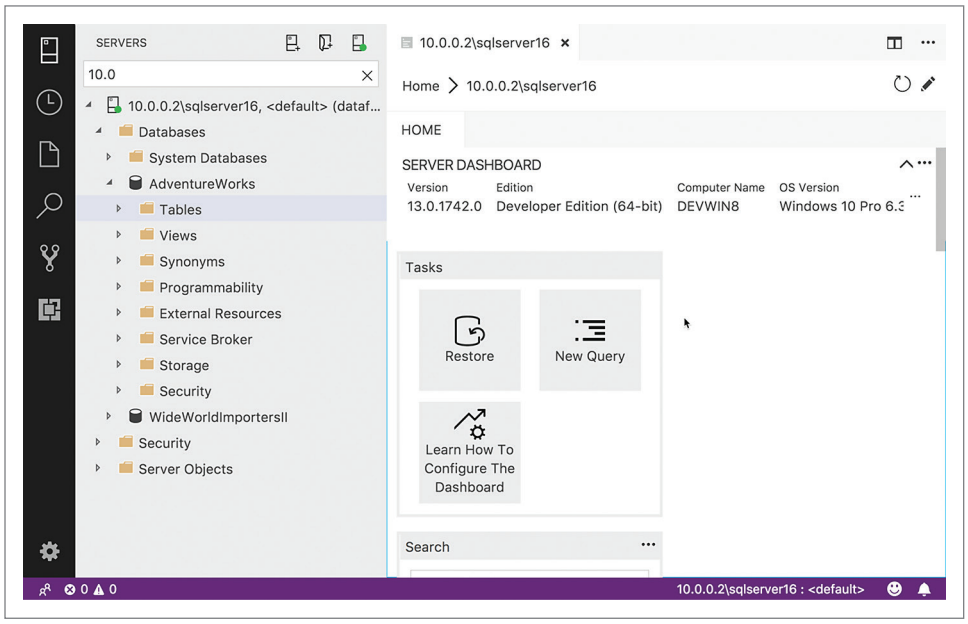


Figure 1 **The Server Explorer Showing a Connected Network SQL Server with Its Default Management Window**

DevExpress®

From Desktops to Web and Mobile

Your Next Great App Starts Here

Experience the DevExpress difference and see why your peers consistently vote our products #1. With our Universal Subscription, you will build your best, see complex software with greater clarity, increase your productivity and create stunning applications for Windows, Web and your Mobile world.

DevExpress Universal ships with 500+ UI controls.
It also includes our royalty-free reporting and dashboard platform.

WIN ASP MVC WPF UWP JS

Download your free 30-day trial today.

devexpress.com/try

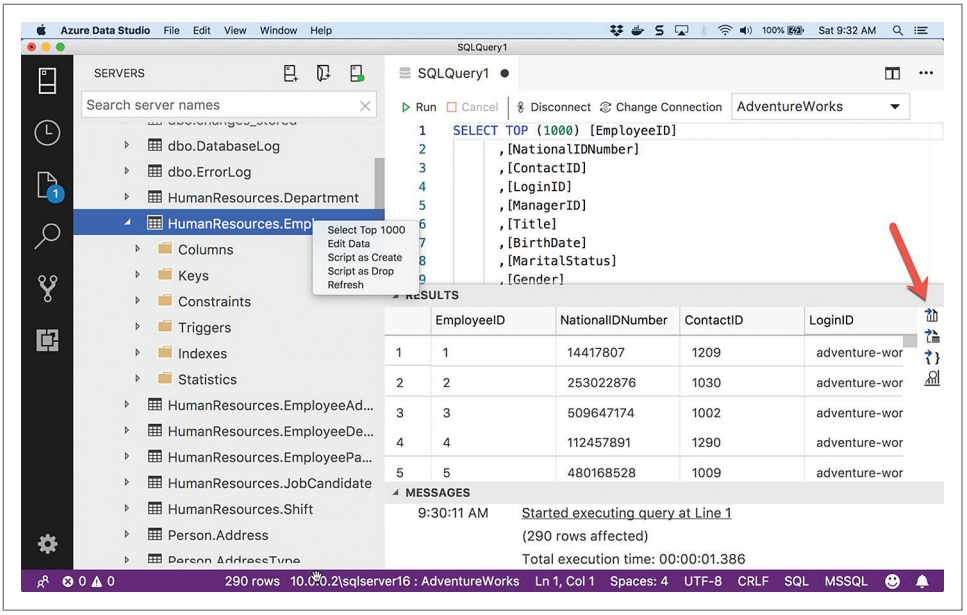


Figure 2 **A Full View of Azure Data Studio While Exporting Selected Rows to CSV**

Azure Data Studio allows you to connect to a variety of SQL Server types. You can connect to an on-premises server on your machine or network, a localdb instance, SQL Server for Linux, SQL Server running in a Docker or Windows container, SQL Azure, or SQL Azure Data Warehouse. There's also preview support for the recently released Azure SQL Database Managed Instances.

I'll start here by creating a new connection to a SQL Server on my network. The new connection wizard provides a connection form where I'll fill in the required information. I'm using SQL authentication and can prompt it to remember my password. After I've connected, the Azure Data Studio Server Explorer lists the available databases (**Figure 1**), with nodes for Tables, Views, Programmability, Security and others you're used to seeing if you use SSMS. You can right-click on a table to view or edit data and build scripts, features that already existed in SQL Operations Studio.

Flat File Imports Using AI

The SQL Server Import extension, which is the result of a hackathon Microsoft held this past summer, is currently in preview and can already import flat data TXT and CSV files. Eventually, it will also be able to import JSON files.

This importing ability is no small feat and is driven by artifical intelligence (AI) using a new technology from Microsoft Research called Program Synthesis Using Examples, or PROSE (microsoft.github.io/prose). As Alan Yu, a SQL Server Program Manager, says, PROSE "can automatically detect a file's data types, delimiters, column names and file structure without the user having to explicitly define the configuration."

Although the export to CSV, JSON or XLSX features aren't new (though their icons are slicker than before), I'm going to start with an export in order to show off the flat file import. And in order to get some data to export, I've used the table context menu's SELECT TOP 1000 option to select all of the rows from the HumanResources.Employee table of the tried-and-true AdventureWorks database. **Figure 2** lets you see the context menu, along with the query pane, results pane and messages pane that resulted from the selection. The display and formatting of all of these panes can be easily configured in the same way you configure VS Code (bit.ly/2lEaFoc). In fact, there are more than 350 customizations you can make in the settings to affect how Azure Data Studio behaves! A red arrow in the image notes the export to CSV icon, which I used to create a file called HREmp.csv. The default settings for exporting CSV files include the column names in the output. With that in hand, I can now show off the flat file import extension that I've already installed.

The extension adds an "Import wizard" menu item to the context menu you get when right-clicking a database in the Server Explorer. As in VS Code, every function has a keyboard command (this one is Ctrl+I) and is accessible from the Command Palette (F1).

The import feature is for importing data into new tables, not existing ones. A form lets you specify the server and database (defaulting to the one you selected earlier), the file to import, and the name and schema for the new table. It then displays a preview of its interpretation of the first 50 rows from the file. The extension makes this look easy, but remember, that magic is thanks to the use of AI by PROSE, which I described earlier.

> The SQL Server Import extension, which is the result of a hackathon Microsoft held this past summer, is currently in preview and can already import flat data TXT and CSV files.

The next step after previewing the data gives you a chance to change how the data is mapped to columns. You can modify the inferred column names and data types, as well as set primary keys and column nullability. While you can't currently do things like remove or add a column, remember that the extension is built within the GitHub repository for the app that I linked to earlier. The team is eager to know what file formats and other related features you'd like to see. You can participate in the discussion at bit.ly/2lXv8wd.

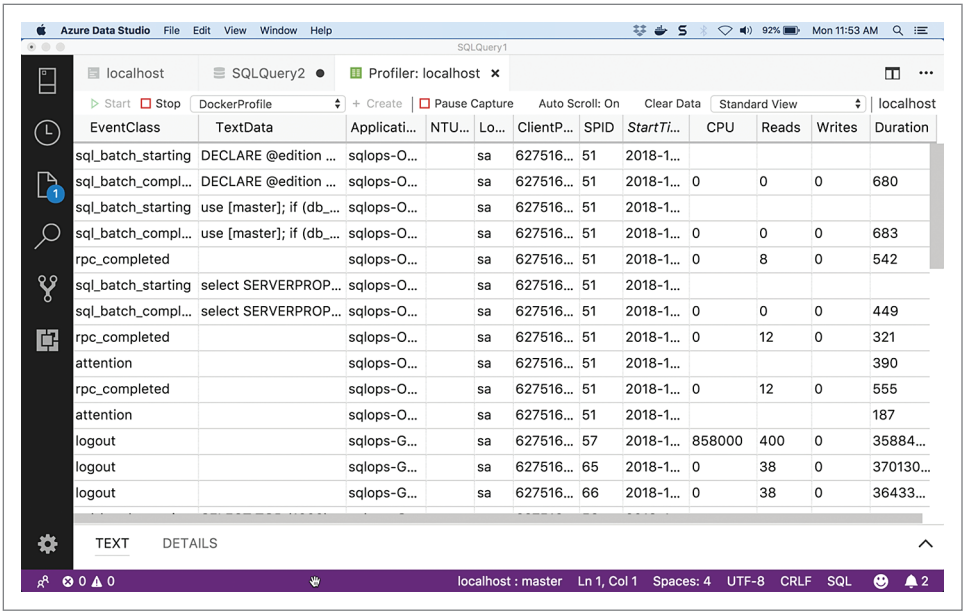


Figure 3 **Standard View Displaying Many Details from All of the Events from a Query**

OMG, Cross-Platform SQL Profiling!

Another extension that's also an early preview as I'm writing this is the SQL Server Profiler extension. Database profiling is a critical step in developing any app. I've made heavy use of the Windows-based SSMS Profiler stand-alone application for decades. Even so, I'm still incredibly clumsy using it—getting the columns I want, applying filters with "magic strings" and more. I don't know if it has ever been updated. I also take advantage of the .NET Core logging capabilities in my apps to see the SQL sent to the database. In fact, my last column was about EF Core logging (msdn.com/magazine/mt830355). But sometimes I really want the database's perspective. And if I'm not on Windows or not on a machine that has the profiler available, I'm at a loss. So, the fact that this extension is part of the cross-platform application feels almost magical to me.

The profiler extension was written using a SQL Server database feature called Extended Events (XEvents). There's also a flavor of XEvents for SQL Azure. You can read more about XEvents at bit.ly/2LfWMoj. The extension's ReadMe indicates the subset of SQL Profiler use cases on which it focuses:

- Stepping through problem queries to find the cause of the problem.
- Finding and diagnosing slow-running queries.
- Capturing the series of Transact-SQL statements that lead to a problem.
- Monitoring the performance of SQL Server to tune workloads.
- Correlating performance counters to diagnose problems.

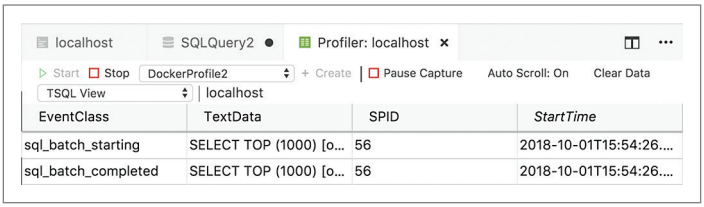


Figure 4 **TSQL View Shows Minimal Info, Only About SQL Events**

Before you can start a profiling session, you need to first connect to a database. Then you can open the profiler extension (Alt+P or Ctrl+Opt+P). You'll be prompted to select a session template and provide a name for the session. Three templates are included with Azure Data Studio: Standard_OnPrem, Standard_Azure or TSQL_OnPrem. The pre-defined standard templates display all events while TSQL displays only the logged TSQL. These templates are defined in the settings, specifying which events are relayed and which filters are applied. You can create your own session templates in the settings, as well.

The profiler will open in its own editor window and automatically start. You can stop and restart the profiler any time using its buttons or a keyboard shortcut—Alt+S (Windows) or Ctrl+Opt+S (Mac).

Initially, when I attempted to start the profiler, I got an error stating "the user does not have permission to perform this action." That's because the SQL Login I had created had limited permissions. Remember, I'm a dev, not a DBA. I fixed that by granting the ALTER TRACE permission to my login. In addition to profiling my network database, I later switched to profiling the SQL Server for Linux database running in a Docker container on the same MacBook where Azure Data Studio was running. (See my article at msdn.com/magazine/mt784660 to learn more about SQL Server in Docker.)

> Database profiling is a critical step in developing an app.

As **Figure 3** shows, standard output will display all events, which, if you've used the SSMS profiler, you know can be very chatty. Once you've created the session you can fine-tune what's displayed with two dropdowns. The first dropdown lets you select from your custom sessions or from three predefined sessions (AlwaysOn_health, system_health and telemetry_xevents) that are built into SQL Server. You can find the details on what's tracked in the system_health session, for example, at bit.ly/2xZcFuP.

Note that you can't change a session when the profiler is running.

The second dropdown provides different views of the captured events: Standard, TSQL, Tuning, TSQL_Locks or TSQL_Duration. Each view is a combination of event filters and the columns that are displayed. For example, the TSQL view displays only EventClass, TextData, SPID and StartTime, as shown in **Figure 4**. The Tuning View shows those four columns, as well as DatabaseID, DatabaseName, ObjectType and LoginName. Like the session templates, the view templates

are also predefined in settings so you can edit the settings to create your own views that will then be available on the dropdown.

There's more to learn about using the profiler, but because it's still an early preview, I expect that by the time you're reading this, there will be new features for you to explore.

Data Science with the SQL Server 2019 Preview Extension

SQL Server 2019, which is currently in preview, has some interesting new features related to Big Data. To go with that, there's an Azure Data Studio extension for SQL Server 2019 (also in preview) that allows you to leverage these features within the IDE. With this combination, Azure Data Studio also becomes a tool for data scientists. SQL Server 2019 offers data clusters with Spark Hadoop Distributed File System (HDFS) clusters, and Azure Data Studio lets you query this data (using SQL) alongside your relational data. SQL Server 2019 also allows you to connect to and virtualize external data, for example, from MongoDB or Oracle. Once connected you can query this data, as well. With Azure Data Studio you can query across both relational and scalable data sources, even joining those resources in your queries. Azure Data Studio also lets you create and use Spark notebooks where you can encapsulate and share your resource connections and queries. These big data features are very new to me and if they are to you, too, I'd highly recommend starting with the five-minute video at bit.ly/2zTgkMI, in which the SQL Server team uses Azure Data Studio and this extension to demonstrate the big data features.

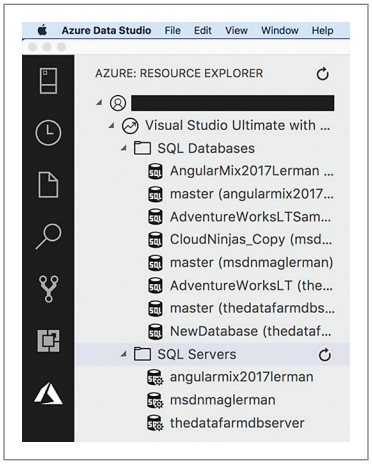


Figure 5 **Browsing Your SQL Azure Resources with the SQL Server 2019 Extension**

> With Azure Data Studio you can query across both relational and scalable data sources, even joining those resources in your queries.

Browse Azure Resources from Azure Data Studio

Azure Data Studio and its predecessor have always been able to connect to SQL Azure. But there's a new feature for browsing your SQL Azure resources and easily connecting to them. This feature started in the SQL Server 2019 extension but is now part of the core application. There's an Azure logo icon on the Activity Bar. The first time you use it you'll be prompted to log in to Azure and allow Azure Data Studio to connect. Then, as shown in **Figure 5**, this view will list all of your accounts and, underneath, all of the Azure SQL Servers and databases within those servers. Then you can just select the database to connect to, which will prepopulate the connection form. All you should need to provide is the password for connecting to the database. In my case, I also had to enable my IP address to connect, but the extension walked me through that. I didn't have to open up the Azure Portal to sort it out.

Speaking of connecting to SQL Azure, when I was working on the previous SQL Operations Studio article, I added a suggestion in GitHub that the team give users a way to copy Azure SQL connection strings from the portal and somehow paste them into Azure Data Studio as another simple way to connect. This is now a feature built into Azure Data Studio, not part of the extension. And it's super cool. If you copy the connection string from the portal, and then in Azure Data Studio click the Create new connection icon (or do that via command palette), the connection form will be prepopulated with all of the details from the connection string. When I did this, the password was populated, even though it wasn't in the connection string I created. Azure Data Studio had remembered that from the previous connection I'd made to another Azure SQL database.

More Extensions

I've highlighted three of the extensions that appealed to me as a developer. But I do want to be sure you're aware of some of the other extensions, some of which will appeal to both devs and DBAs. RedGate created a SQL Search extension. Microsoft built an extension around Adam Machanic's sp_whoisactive activity monitoring tools. It also created extensions for SQL Server Agent, Azure SQL Warehouse Data Insights and Server Reports, which includes, for example, database space and buffer usage, CPU utilization and wait counts. There are extensions from MVPs for gaining more insights and merging scripts. And if you're a keyboard-shortcut fan, you'll be interested in the extension to import popular keymaps from SSMS into Azure Data Studio that was created by Kevin Cunnane from the SQL Server tools team.

If you're interested in building extensions, you can find help in the documentation at bit.ly/2zTkROR.

I now have Azure Data Studio installed on every single computer on which I work. Desktops and laptops, Windows and macOS. I even have it on the Windows desktop computer where I have full-blown SQL Server and SSMS installed and find that I'm reaching for Azure Data Studio on that machine much more frequently than SSMS because of the nature of the tasks I am typically performing. ■

Julie Lerman *is a Microsoft Regional Director, Microsoft MVP, software team coach and consultant who lives in the hills of Vermont. You can find her presenting on data access and other topics at user groups and conferences around the world. She blogs at the thedatafarm.com/blog and is the author of "Programming Entity Framework," as well as a Code First and a DbContext edition, all from O'Reilly Media. Follow her on Twitter: @julielerman and see her Pluralsight courses at juliel.me/PS-Videos.*

Thanks to the following Microsoft technical expert for reviewing this article: *Alan Yu*

File Format APIs

Open, Create, Convert, Print and Save files from your applications!

Try risk free - 30 day trial

ASPOSE

Download a Free Trial at

https://downloads.aspose.com

Aspose.Words

Create, edit, convert or print Word documents (DOC, DOCX, RTF etc.) in your .NET, Java and Android applications.

Aspose.Cells

Develop high performance .NET, Java and Android applications to Create, Edit or Convert Excel worksheets (XLS, XLSX, ODS etc).

Aspose.Pdf

Manipulate PDF file formats (PDF, PDF/A, XPS etc.) using our native APIs for .NET, Java and Android platforms.

Aspose.Slides

Create, edit or convert PowerPoint presentations (PPT, PPTX, ODP etc.) in your .NET, Java and Android applications.

Aspose.Email

Create, Edit or Convert Outlook Email file formats (MSG, PST, EML etc.) and popular network protocols.

Aspose.BarCode

Generate or recognize barcodes (Code128, PDF417, Postnet etc.) using our native APIs for .NET and Java.

Aspose.Imaging

Deliver efficient applications to Create, Draw, Manipulate or Convert image file formats.

Aspose.Tasks

Develop high performance apps to Create, Edit or Convert Microsoft Project® document formats.

- Aspose.Diagram
- Aspose.Note
- Aspose.3D
- Aspose.CAD
- Aspose.HTML
- Aspose.GIS

Americas: +1 903 306 1676 EMEA: +44 141 628 8900 Oceania: +61 2 8006 6987
sales@asposeptyltd.com

TED NEWARD

How To Be MEAN: End-To-End

Welcome back again, MEANers.

In the last piece, I talked about unit testing, which verifies that the code on the client is working correctly. However, most of the time unit tests look to mock out the back-end interaction, in favor of creating a more deterministic set of test results; that's useful, but not sufficient, much of the time. Sometimes, you really need to make sure the front end is communicating effectively with the back end, particularly when both are in flux—changes to the JSON exchanged between the two, for example, can each be unit-tested as correct within themselves, but once you start actually passing data across the wire, they can disagree, and break. Nobody wants that, particularly since (as every developer with more than three years' experience can tell you) it will only happen when you're doing a demo in front of somebody much higher in the org chart than you.

This is the province of Angular's end-to-end (e2e) testing, and where unit testing looks to test each component individually and, isolated away from the rest of the system, e2e testing looks to test the system as a whole. This makes e2e testing more brittle, since it depends on a lot more factors, but it's also just as important as unit testing in the long run.

Back to the Beginnings

As with the unit-testing discussion, Angular tries to make sure that testing is available from the beginning of your greenfield application by building some basic scaffolding in place when the application is first generated (assuming you use the Angular command-line interface [CLI] tool). The e2e directory, off the root of the project, contains the necessary code to run an end-to-end test, so I'm going to give it a spin before doing anything else: "npm run e2e" launches the end-to-end testing, and unlike the unit-testing tooling, e2e runs through a test pass, then halts. The intent here is that the unit-testing tooling should be running near-constantly during development to verify that things are working "in the small," and the end-to-end tests should be run only when it's time to verify that no regressions popped up.

The default test is just to verify the scaffolded app's "Welcome to app!" message, so one easy first change is to modify the AppComponent's title property (remember, from way back when we first started, that property is in app.component.ts) to "SpeakerApp" and check to see that it shows up. (Don't forget to change the unit test in app.component.spec.ts, too, by the way.) Make that change, make sure the unit tests pass, then run "npm run e2e" again and note the failure. Well, try to note the failure—the browser closes down but the console window in which you ran "npm" will highlight (in red) the test failure.

So, the next step, then, is to modify the end-to-end tests to reflect the change you've made to the code.

Of Pages and Objects

End-to-end testing uses the ProtractorJS framework (at protractortest.org) to run the browser in an "automated" mode, and the code to drive Protractor lives in the app.po.ts file. The "ts" suffix should be pretty familiar by now, but the "po" stands for *page object*, which is a common idiom in the Web end-to-end testing world. A page object is, literally, an object that represents all the actions possible on a given Web page, so the app.po.ts page would represent the application's home page, according to the naming convention. Sure enough, if you have a look at it in the editor, you see:

```
import { browser, by, element } from 'protractor';

export class AppPage {
  navigateTo() {
    return browser.get('/');
  }

  getParagraphText() {
    return element(by.css('app-root h1')).getText();
  }
}
```

Note that the code here isn't doing any testing—this is literally a bundle of code that serves a utility role for the end-to-end tests, and nothing more. The navigateTo method simplifies bringing this page up in the browser, and the getParagraphText method simply extracts the text for the title by using a CSS selector to find the H1 element out of the DOM that corresponds to the app-root component. As a matter of fact, that's probably a misnamed method—it really should be called getTitleText, so let's make that change and do the corresponding change in the actual test code. (Note that if you monkey around with the browser's title, via the <title> HTML tag, you might want to rethink the method name again, but because this isn't a production app, I'm not going to stress over it.)

Now open the other file in the e2e directory, app.e2e-spec.ts:

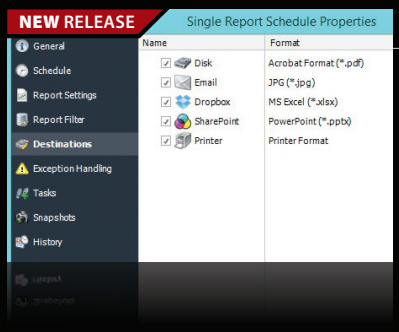
```
import { AppPage } from './app.po';

describe('full-app App', () => {
  let page: AppPage;

  beforeEach(() => {
    page = new AppPage();
  });

  it('should display welcome message', () => {
    page.navigateTo();
    expect(page.getParagrahText()).toEqual('Welcome to app!');
  });
});
```

Here you can see the usual Jasmine test format, with the describe method and it functions, and notice how the test fires up



PBRS (Power BI Reports Scheduler) | from $8,132.21

Christiansteven

Schedules & delivers Power BI reports to unlimited recipients with one Power BI License.

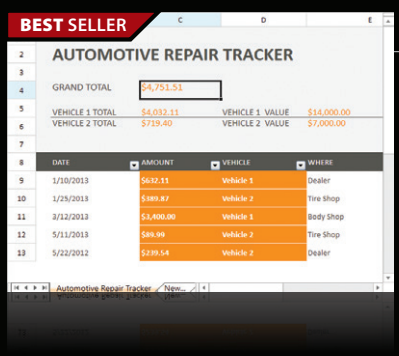
- Exports reports to PDF, Excel, Excel Data, Word, PowerPoint, CSV, JPG, HTML, PNG and ePub
- Sends reports to email, printer, fax, folder, FTP, DropBox and SharePoint
- Uses database queries to automatically populate report filters, email addresses & body text
- Adds flexibility with custom calendars e.g. 4-4-5, holidays, "nth" day of the month, etc.
- Responds instantly by firing off reports when an event occurs e.g. database record is updated

DevExpress DXperience 18.2 | from $1,439.99

DevExpress

A comprehensive suite of .NET controls and UI libraries for all major Microsoft dev platforms.

- WinForms – New Sunburst Chart, Office Navigation UX, SVG Office 2019 skins
- WPF – New Gantt control, improved Data Filtering UX and App Theme Designer
- ASP.NET & MVC – New Adaptive Layouts, improved Rich Text Editor and Spreadsheet
- Reporting – Vertical Band support, Free-hand drawing and improved Report wizards
- JavaScript – New HTML/Markdown WYSIWYG editor, Improved Grid and TreeList performance

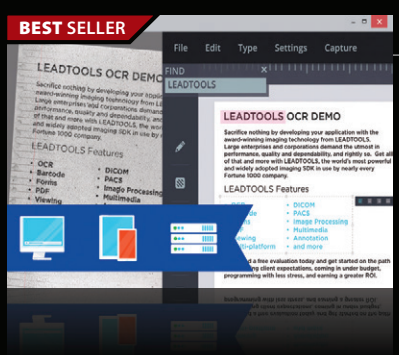


Spread.NET | from $1,476.52

GrapeCity

Deliver multi-functional spreadsheets in less time with Visual Studio.

- Designed from the ground up for maximum performance and speed
- Gives you the features you expect for seamless Excel compatibility with NO Excel dependency
- Powerful calculation engine handles huge numbers of rows and columns with ease
- Use the extensive API to control every aspect of workbooks, worksheets, ranges, and cells
- Benefit from the integrated calculation engine with more than 462 Excel functions available



LEADTOOLS Document Imaging SDKs V20 | from $2,995.00 SRP

LEADTOOLS

Add powerful document imaging functionality to desktop, tablet, mobile & web applications.

- Universal document viewer & conversion framework for PDF, Office, CAD, TIFF & more
- OCR, MICR, OMR, ICR and Forms Recognition supporting structured & unstructured forms
- PDF SDK with text edit, hyperlinks, bookmarks, digital signature, forms, metadata
- Barcode Detect, Read, Write for UPC, EAN, Code 39, Code 128, QR, Data Matrix, PDF417
- Zero-footprint HTML5/JavaScript UI Controls & Web Services

We accept purchase orders.
Contact us to apply for a credit account.

USA
ComponentSource
650 Claremore Prof Way
Suite 100
Woodstock
GA 30188-5188
USA

Europe (UK)
ComponentSource
The White Building
33 King's Road
Reading, Berkshire
RG1 3AR
United Kingdom

Europe (Ireland)
ComponentSource
Unit 1E & 1F
Core B, Block 71
The Plaza, Park West
Dublin 12
Ireland

Asia Pacific
ComponentSource
7F Kojimachi Ichihara Bldg
1-1-8 Hirakawa-cho
Chiyoda-ku
Tokyo, 102-0093
Japan

Sales Hotline - US & Canada:
(888) 850-9911
www.componentsource.com

GSA Schedule Contract GS-35F-0188R

an AppPage instance, navigates to it, then tests to see if the paragraph text matches the expectation. I know there are two changes I need to make—one to use the renamed getTitleText method, and the other to change the text to "Welcome to SpeakerApp," in order to reflect the change in the title. Once those changes are made and "npm run e2e" is invoked, everything's green again.

Protractor 101

The Protractor Web site has the complete documentation set, but fortunately most of the Protractor API is pretty self-documenting, at least from a high-level perspective. The Protractor home page has an example of a relatively simple end-to-end test that exercises a "to-do" list application, demonstrating how to find elements on the page by using either CSS selectors (element(by.css(…))), model object (element(by.model(…))), or even grabbing a collection of elements via an Angular repeater (element.all(by.repeater(…))), which can then be counted and individually examined, as desired. The Protractor tutorial also mentions that elements can be obtained by HTML identifier (by.id(…)) and by binding, meaning the element is bound to a given variable (by.binding(…)); these are all collectively known as ElementFinders in the Protractor documentation, in case more details are needed.

Once the e2e tests are set up and running, however (and the default Angular CLI scaffolding gives a strong leg up on that, given that it provides a starting point), really, writing e2e tests is about the same as writing any other tests: arrange, act, assert. Given that this is a Javascript-based world, of course you can change the testing framework you use if you so choose, but assuming you work with the same tools out of the box that the CLI establishes, you're already on your way towards writing a comprehensive set of e2e tests.

One thing that bears mentioning: whether or not you use page objects is entirely your personal choice, but, frankly, given that e2e tests tend to be more "macro" tests (meaning, you typically don't test just one page or component, but a whole series of steps) and/or user-acceptance-style tests, it's generally helpful to have page objects in place to make the tests easier to read and—particularly—maintain. Remember, UI tends to be one of those things that shifts often during user demos, and the page object approach can help minimize the "churn" that can happen to e2e tests because of user changes.

Incorporating Server Bits

Given that I suggested that running the end-to-end tests means using the server, and that running the e2e tests should be a pretty common thing, it's probably a good idea to make sure that running the e2e tests also fires up the server. Otherwise, humans have to remember to run something before running the e2e tests, and that's usually a recipe for frustration in the long term. One way to do this, of course, is to write a batch file or PowerShell script to run both; another approach is to simply edit the package.json that Angular generated to have the "e2e" part of the file issue the right shell commands to launch a local version of the API server. (It's not a flawless approach, but it serves for now.) That way, you remain consistent with the Angular CLI conventions.

Assuming that the server parts are in a subdirectory called "server" (and that you're using the LoopbackJS-based server I wrote back in October, 2017 (msdn.com/magazine/mt826349), then all you need to do is get "npm run" to issue a "node server" command, which will drop into the server directory and issue a "node .," which is enough to launch the Loopback bits:

```
{
  "name": "full-app",
  // ...
  "scripts": {
    "ng": "ng",
    "start": "ng server",
    "build": "ng build --prod",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "start node server & ng e2e"
  },
  // ...
}
```

The "start" in front of the "node server" is to tell the shell to spin off node server into a separate command window; this will have the unfortunate side effect of leaving the server window running after the e2e tests finish, but it suffices for now. (If this really creates heartache, replace the two commands with a batch file or PowerShell script, and then you're in best-of-both-worlds territory.) Now, the server is up and running, and the client can start making calls to it.

Arguably, for this to be a true end-to-end test, the e2e tests should be operating against a shared server, but then you're back to the problem of non-determinism in the tests. Others may disagree, but I prefer a local server over a remote/shared one for this sort of testing.

Wrapping Up

By the way, although it realistically shouldn't be a concern by this point, as of this writing, the Angular team just released Angular 7. This means this column has seen at least six significant releases of Angular since its start, and given that none have created any major rewrites on our part, that should give us a certain comfort level in adopting Angular for any long-lived application and, hopefully, addresses a few of the concerns toward adopting Angular in your company or team.

On a logistical note, this column marks the 30th article written on the MEAN stack, and while there's clearly more I could delve into, the name of the column is "Working Programmer," not "MEAN Programmer," and it's time to move on. Angular's cool and all, but this could go on ad *infinitum*, and there are lots of other interesting things out there to explore: new C# features, new F# features, and maybe I'll even take a dive into the inner workings of the CLR, or a JavaScript engine (either V8, the one that ships with NodeJS, or Chakra, Microsoft's version of V8), for a bit. Before ending, though, let's be gracious consumers, and say, thanks, Angular, for all the good memories over the last few years, and we'll be seeing you around. Keep up the good work, and we'll drink a toast to you when you hit Angular 10 and 15 and 20 and so on. For now, though, fare well and smooth sailing.

As always, happy coding! ■

Ted Neward *is a Seattle-based polytechnology consultant, speaker, and mentor, currently working as a director of Engineering and the director of Developer Relations at Smartsheet.com. He's written a ton of articles, authored and co-authored a dozen books, and speaks all over the world. Reach him at ted@tedneward.com or read his blog at blogs.tedneward.com.*

Thanks to the following technical expert for reviewing this article: *Garvice Eakins (Smartsheet.com)*

DocuVieware

Universal HTML5 and Document Management Kit

Easy integration

Full support for custom snap-in

Zero-footprint solution

Fully customizable UI

Mobile devices optimization

Fast & crystal-clear rendering

Check the New Features and the Online Demos
60-day Free Trial Support Included at **www.docuvieware.com**

Powered by **GdPicture**.NET

DocuVieware is an ORPALIS imaging technologies product

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

Choose VSLive! For:

- In-depth developer training
- Unparalleled networking
- World-class speakers
- Exciting city adventures

Developer Training Conferences and Events

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360 An 1105 MEDIA company

JOIN US IN 2019!

March 3-8, 2019

Bally's Hotel & Casino

Visual Studio Live! kicks off 2019 in the heart of Las Vegas with 6 days of hard-hitting Hands-On Labs, workshops, 60+ sessions, expert speakers and several networking opportunities included! Register to join us today!

REGISTER NOW!
vslive.com/lasvegas

April 22-26, 2019

Hyatt Regency

For the first time in our 20-year history, Visual Studio Live! is heading down south to New Orleans for intense developer training, bringing our hard-hitting sessions, well-known coding experts and unparalleled networking to the Big Easy!

REGISTER NOW!
vslive.com/neworleans

June 9-13, 2019

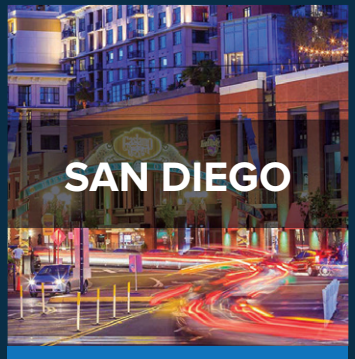
Hyatt Regency Cambridge

Join Visual Studio Live! for an amazing view of Beantown, bringin g our infamous speakers for intense developer training, Hands-On Labs, workshops, sessions and networking adventures to the Northeast.

August 12-16, 2019

Microsoft HQ

Join our Visual Studio Live! experts at the Mothership for 5 days of developer training and special Microsoft perks unique to our other show locations. Plus, we are adding the ever-so popular full-day Hands-On Labs to the agenda in Redmond for the first time this year!



September 29, 2019

Westin Gas Lamp

Head to the heart of the San Diego Gaslamp District with Visual Studio Live! this Fall as we immerse ourselves with all things for developers, including several workshops, sessions and networking opportunities to choose from.

October 6-10, 2019

Swissotel

Head to the Windy City and join Visual Studio Live! this October for 5 days of unbiased, developer training and bringing our well-known Hands-On Labs to the city for the first time.

November 17-22, 2019

Royal Pacific Resort at Universal

Visual Studio Live! Orlando is a part of Live! 360, uniquely offering you 6 co-located conferences for one great price! Stay ahead of the current trends and advance your career – join us for our last conference of the year!

NEW IN 2019!

On-Demand Session Recordings for One Year!

Get access to all sessions (not including Hands-On Labs or Workshops) at each show for a year. Learn more at vslive.com.

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

vslive.com #VSLIVE

FRANK LA VIGNE

Market Basket Analysis

There is a bit of artificial intelligence (AI) that you have, no doubt, encountered, especially as the holiday shopping season is underway—the recommended purchase. Nearly every online retailer will display additional product recommendations, sometimes under the header of "Frequently bought together," or "Customers who purchased X also purchased Y." According to one study by McKinsey in 2013 (bit.ly/2yK3Bu8), 35 percent of what consumers purchase on Amazon come from product recommendation algorithms. What's more, this tactic is no longer limited to retailers, online streaming services like Netflix and YouTube use sophisticated recommendation algorithms to keep viewers tuned in longer.

Clearly, recommendation systems have an impact on our daily lives. You could argue that they're the most prominent form of AI that consumers encounter. In this column, I'll explore a basic form of recommendation system known as a Market Basket Analysis.

Market Basket Analysis

Market Basket Analysis, also known as Affinity Analysis, is a modeling technique based on the theory that if you buy a certain group of items, you're more likely to purchase another group of items. For example, someone purchasing peanut butter and bread is far more likely to also want to purchase jelly. However, not all relationships are as immediately obvious. Foreknowledge of consumer behavior can increase sales and give the retailer a significant edge against competitors. Strictly speaking, Market Basket Analysis is just one application of association analysis techniques, although many online articles and tutorials may confuse the two. To put it in perspective of other machine learning techniques I've written about before, Market Basket Analysis is an unsupervised learning tool that requires little in the way of feature engineering and a limited amount of data cleaning and preparation. In practice, insights gleaned from Market Basket Analysis can be further explored with other AI or data science tools.

Despite its ability to uncover hidden patterns, Market Basket Analysis is relatively easy to explain and doesn't require knowledge of advanced statistics or calculus. However, there are a few terms and conventional notations to review. First, the notions of cause and effect are referred to as **antecedent** and **consequent**. In the example I mentioned previously, peanut butter and bread are the antecedent and jelly is the consequent. The formal notation for this relationship would be {Peanut Butter, Bread} -> {Jelly} indicating that there's a connection between these items. Also take note that both antecedents and consequents can consist of multiple items.

There are three important mathematical measures required for Market Basket Analysis: Support, Lift and Confidence. Support represents the number of times antecedents appear together in the data. To simplify the example, imagine the following relationship: {Peanut Butter} -> {Grape Jelly}. Given 100 customers (and one transaction per customer), consider the following scenario:

- 15 customers bought Peanut Butter
- 13 bought Grape Jelly
- 11 bought Peanut Butter and Grape Jelly

Despite its ability to uncover hidden patterns, Market Basket Analysis is relatively easy to explain and doesn't require knowledge of advanced statistics or calculus.

Support represents the number of times items appear in a transaction together, which in this example is 11 out of 100, or 0.11. To use statistical terms, there's a probability of 11 percent that any given transaction will include both Peanut Butter and Grape Jelly. Confidence takes the value of Support (.11) and divides it by the probability of a transaction of having Grape Jelly alone, equating to a value of 0.846. This means that nearly 85 percent of the time that Grape Jelly was purchased, it was purchased along with Peanut Butter. Finally, there's Lift, which takes Confidence (0.846) and divides it by the probability of Peanut Butter. This equate to 5.64 (rounded to two decimal places).

All this might be clearer in a simple chart, as shown in **Figure 1**.

Market Basket Analysis in Action

Keeping the previous metrics in mind, it's time to try out Market Basket Analysis on a real data set. The first step is to get retail data to analyze. Fortunately, the University of California, Irvine, provides a dataset that contains transactions for a U.K.-based Web

Code download available at bit.ly/2OhBJCR.

Figure 1 **Support Confidence and Lift Values**

Measure	Formula	Value
Support	P(Peanut Butter & Grape Jelly)	.011
Confidence	Support / P(Grape Jelly)	0.846
Lift	Confidence / P(Peanut Butter)	5.64 (rounded)

site. More information about the dataset is available at bit.ly/2DgATFI. Create a Python 3 notebook on your preferred platform (I covered Jupyter notebooks in a previous column at msdn.com/magazine/mt829269). Create an empty cell, enter the following code to download the sample data, and execute the cell:

```
! curl http://archive.ics.uci.edu/ml/machine-learning-databases/00352/
Online%20Retail.xlsx -o retail.xlsx
```

> While Market Basket Analysis doesn't require rigorous data munging, it does make sense to remove extraneous records, such as those with null invoice numbers and canceled orders.

Once this completes, enter the following code into a new cell to load the Excel spreadsheet into a Pandas DataFrame and print out the columns of the data set:

```
import pandas as pd
df = pd.read_excel('retail.xlsx')
print( df.columns)
```

The output will look something like this:

```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

While Market Basket Analysis doesn't require rigorous data munging, it does make sense to remove extraneous records, such as those with null invoice numbers and canceled orders. It's also useful to remove extraneous spaces in the product descriptions and convert all the invoice numbers to string. You can do that by executing the following code:

```
df['Description'] = df['Description'].str.strip()
df.dropna(axis = 0, subset=['InvoiceNo'], inplace = True)
df['InvoiceNo'] = df['InvoiceNo'].astype('str')
df = df[~df['InvoiceNo'].str.contains('C')]
```

Now enter the following code to aggregate the data and view it from a country level:

```
df.groupby('Country').count().reset_index().sort_values(
  'InvoiceNo', ascending = False).head()
```

Next, I'll rearrange the data with each product one hot encoded and one transaction per row. One hot encoding is a data transformation technique where categorical values are converted into columns, with the value of 1 entered where a categorical value is present. I will also limit the scope of the dataset to one country, in this case France, to compare consumer behavior in an individual market. Enter and execute the following code to do that (notice the shape of the basket_uk data frame in the cell output; the one hot encoding expands the columns from 8 to 4175):

```
basket_fr = (df[df['Country']=="France"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))

basket_fr.head(10)
```

A quick glance at the results reveals an issue with my one hot encoding. The sixth item down has a value of 24.0 in the second column. My intention was to have either a one or a zero in the data, not quantities. Therefore, I will need to locate any non-zero values and convert them to 1. To fix this, run the following code and note that the 24.0 has been converted to a 1:

```
def sum_to_boolean(x):
  if x<=0:
    return 0
  else:
    return 1

basket_fr_final = basket_fr.applymap(sum_to_boolean)
basket_fr_final.head(10)
```

I will use MLXTEND (rasbt.github.io/mlxtend) to further analyze the data. MLXTEND is a Python library of useful tools for common data science tasks, including Market Basket Analysis. To install this library from within the notebook, execute the following code:

```
! pip install mlxtend
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
6	(CHILDRENS CUTLERY SPACEBOY)	(CHILDRENS CUTLERY DOLLY GIRL)	0.068878	0.071429	0.063776	0.925926	12.962963	0.058856	12.535714
7	(CHILDRENS CUTLERY DOLLY GIRL)	(CHILDRENS CUTLERY SPACEBOY)	0.071429	0.068878	0.063776	0.892857	12.962963	0.058856	8.690476
39	(ALARM CLOCK BAKELIKE RED)	(ALARM CLOCK BAKELIKE PINK, ALARM CLOCK BAKELI...	0.094388	0.073980	0.063776	0.675676	9.133271	0.056793	2.855230
34	(ALARM CLOCK BAKELIKE PINK, ALARM CLOCK BAKELI...	(ALARM CLOCK BAKELIKE RED)	0.073980	0.094388	0.063776	0.862069	9.133271	0.056793	6.565689
38	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE PINK, ALARM CLOCK BAKELI...	0.096939	0.073980	0.063776	0.657895	8.892922	0.056604	2.706829
35	(ALARM CLOCK BAKELIKE PINK, ALARM CLOCK BAKELI...	(ALARM CLOCK BAKELIKE GREEN)	0.073980	0.096939	0.063776	0.862069	8.892922	0.056604	6.547194
2	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE RED)	0.096939	0.094388	0.079082	0.815789	8.642959	0.069932	4.916181

Figure 2 **Association Rules from French Customers**

With the MLXTEND package installed, it's time to import the relevant libraries from MLXTEND, like this:

```
from mlxtend.frequent_patterns import association_rules
from mlxtend.frequent_patterns import apriori
```

In a new cell, enter the following code to view sets of items with at least 6 percent support:

```
frequent_itemsets_fr = apriori(basket_fr_final, min_support = 0.06,
  use_colnames = True)
frequent_itemsets_fr.sort_values('support', ascending = False).head()
```

With the key sets of items identified, I can now apply the association rules library to discover customers' purchase behaviors. Enter the following code and execute it:

```
a_rules = association_rules(frequent_itemsets_fr, metric = "lift",
  min_threshold = 1)
a_rules.sort_values('lift',ascending = False)
```

The results, sorted by Lift, should look similar to those in **Figure 2**, revealing the purchasing patterns of French customers on the site. A quick glance at the results shows that customers who buy CHILDRENS CUTLERY SPACEBOY also purchase CHILDRENS CUTLERY DOLLY GIRL, and that customers who buy an alarm clock in one color also purchase an alarm clock in another color. As far as actionable insights go, I could recommend to the site owners to offer bundle deals on gender-specific cutlery, as well as offer multi-color alarm clock bundles.

However, keep in mind that this list is sorted by Lift and not occurrence. It may not make sense to introduce a new bundle or product offering if it isn't popular enough. To get a view of how popular these cutlery items are, enter the following Python code:

```
print( basket_fr_final['CHILDRENS CUTLERY SPACEBOY'].sum())
print( basket_fr_final['CHILDRENS CUTLERY DOLLY GIRL'].sum())
```

The results aren't promising; only 27 for SPACEBOY and 28 for DOLLY GIRL. With a little exploration, I find an association rule with some more promise. It turns out that the association rule index 50 (SET/20 RED RETROSPOT PAPER NAPKINS) is the antecedent for red paper cups and red paper plates. Enter the following code to see how many units are sold:

```
basket_fr_final['SET/20 RED RETROSPOT PAPER NAPKINS'].sum()
```

While the number is low, it stands to reason that customers purchasing disposable cups want matching paper plates and napkins. A savvy retailer could easily package these into a bundle offer to entice the customer to purchase.

> Sharp-eyed readers will notice that there are two other metrics in the resulting data frame: Leverage and Conviction.

Sharp-eyed readers will notice that there are two other metrics in the resulting data frame: Leverage and Conviction. These are additional values to be considered when performing Market Basket Analysis. More information about this can be found by exploring so-called "alternative measures of interestingness." Wikipedia is a handy place to start (bit.ly/2AECRNh).

Recall that when I aggregated the data from a country level, there were vastly more invoices from the United Kingdom than any other country. Perhaps more could be learned by examining customer behavior with more raw data available. Let's explore this by entering the following code into a new cell and executing it, like so:

```
basket_uk = (df[df['Country']=="United Kingdom"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))

basket_final_uk = basket_uk.applymap(sum_to_boolean)
frequent_itemsets_uk = apriori(basket_final_de, min_support = 0.06,
  use_colnames = True)

a_rules_uk = association_rules(frequent_itemsets_uk, metric = "lift",
  min_threshold = 1)
a_rules_uk.sort_values('lift',ascending = False).head()
```

> Market Basket Analysis provides a great entry point for persons and organizations looking to explore data science.

For the United Kingdom, the execution time is much longer due to the larger dataset. Also note that the results are quite different. Could this be a result of more data changing the analysis, or simply a function of different customer preferences in a different market? Or could this be that the retailer offers different products for sale in different markets? These are the kinds of variables you have to understand as you work through your analysis. In this case, we have little context beyond this being an online retailer based in the United Kingdom. However, in a real-world project, engagement with the business' subject-matter experts is a critical element for success in data analytics projects.

Wrapping Up

In this article, I explored the use of Market Basket Analysis to uncover patterns in consumer behavior. Market Basket Analysis belongs to the larger field of Affinity Analysis, which major companies use to get customers to spend more money on products and more time on streaming platforms.

Market Basket Analysis provides a great entry point for persons and organizations looking to explore data science. The barrier to entry is low in terms of the mathematical skill. In fact, the mathematics doesn't go beyond simple division and basic probability theory. It's an easy problem space to explore for beginners and offers a great place to start in applying AI to the enterprise. That said, do not be fooled into thinking that this isn't a powerful means to conduct data science or show value to company leadership. The impacts on the bottom line can be significant. ■

Frank La Vigne *works at Microsoft as an AI Technology Solutions professional where he helps companies achieve more by getting the most out of their data with analytics and AI. He also co-hosts the DataDriven podcast. He blogs regularly at FranksWorld.com and you can watch him on his YouTube channel, "Frank's World TV" (FranksWorld.TV).*

Thanks to the following technical expert for reviewing this article: *Andy Leonard (Enterprise Data & Analytics)*

Spreadsheets Everywhere.

SpreadsheetGear 2017 Released

SpreadsheetGear 2017 adds a new SpreadsheetGear for .NET Standard product, official support for Excel 2013 and Excel 2016, 51 new Excel functions for a total of 449 fully supported functions, full conditional formatting support, enhanced workbook protection and encryption, cell gradient rendering and more.

Support for iOS, Android, Linux, macOS, UWP and more

SpreadsheetGear for .NET Standard enables cross-platform developers to enjoy the same high performance Excel-compatible reporting, charting, calculations and more relied on by thousands of Windows developers for 10+ years.

Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application using spreadsheet technology built from the ground up for performance, scalability and reliability.

WIN Windows Forms | Silverlight | WPF

Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.

Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.

Fastest Calculations

Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

Toll Free USA (888) 774-3273 | Phone (913) 390-4797 | sales@spreadsheetgear.com

Gain Insight from Conversations Using Process Mining with LUIS

Zvi Topol

The Language Understanding Intelligence Service (LUIS) is a Microsoft Cognitive Services API that offers natural language understanding as a service for developers. As part of that service, when given user input, called an utterance, LUIS returns the intent detected behind the utterance, that is, what the user intends to ask about. In a previous article (msdn.com/magazine/mt847187), I discussed how to improve LUIS intent detection by leveraging open source tools that provide word-level insights into how intents are classified.

In this article, I'm going to take a step or two up the hierarchy and move from words to entire conversations. In particular, I'll focus on how to get insights from conversational data, by which I mean data that's composed of sequences of utterances that collectively make a conversation.

To that end, I'll introduce process mining, a field of analytics specializing in deriving insights from sequences of events. I will then show how to transform conversational data, which is innately unstructured, into a structured dataset by applying LUIS to each utterance in a conversation. Finally, I'll use process mining on the transformed, structured dataset to derive insights about the original conversations.

Let's get started.

This article discusses:

- Overview of process mining
- Getting conversational data ready for process mining
- Applying process mining to conversational data using Disco

Technologies discussed:

Microsoft Cognitive Services, LUIS, Disco

Code download available at:

msdn.com/magazine/1218magcode

Process Mining

Process mining is a field of data analytics whose primary focus is the discovery of processes (sequences of events or states with a specific outcome) and the delivery of insights about such processes from event log data. Event log data is a dataset that specifies for each event the timestamp at which it occurred, along with other possible fields or attributes describing the event. (More on that later.)

Domains to which process-mining tools have been applied include financial services, health care and manufacturing. Examples include mortgage application process analysis to identify bottlenecks and delays in customer applications, patient status monitoring through the different departments/wards, all the way from patient admission through to release, and part-production process improvement on a manufacturing machine line.

Scaling the algorithms that power process mining such that they are able to process large datasets of events has been a critical area of focus in both academia and industry. Another challenge hampering the large-scale adoption of process mining is the fact that in many organizations, data isn't currently collected and formatted in a way that it can be properly consumed by process-mining software. I expect this to change as more organizations become aware of the value of digital transformation and data-driven decision making.

Process mining has been an active research area for many years and, therefore, has benefited from the development of many algorithms that enable the discovery of processes and the deriving of insights. In this context, it's worth mentioning Prof. Wil van der Aalst (bit.ly/2QNBvEW), currently at Aachen University, who is considered one of the founding fathers of this field. Prof. van der Aalst is also the author of "Process Mining: Data Science in Action" (Springer 2016), one of the important textbooks in the field and a good starting point if you'd like to deepen your knowledge about process-mining algorithms.

As far as mature commercial process-mining software products go, there are a number of companies with useful offerings, including Celonis (celonis.com), Fluxicon (fluxicon.com) and Minit (minit.io). There are also a few open source products, in particular ProM (promtools.org/doku.php), which offers integration with data mining tools such as RapidMiner (rapidprom.org).

In addition to standalone process-mining products, you'll also find packages and libraries available for popular data analysis languages such as R and Python that provide programmatic interfaces to process-mining functionality. Some examples include edaR—exploratory and descriptive event-based data analysis in R (bit.ly/2OSGozv) and PMLab, which is an interactive environment written in Python that includes building blocks for programmatically applying process-mining techniques (pmlab/pmlab-full).

In this article, I'll use Disco as the primary process-mining tool. You can download a trial version from fluxicon.com/disco. Disco has an intuitive UI, is easy to use and has a comprehensive set of features. It's also a popular tool in various online introductory classes on process mining.

> Event log data is a dataset that specifies for each event the timestamp at which it occurred, along with other possible fields or attributes describing the event.

While I'll show how to derive insights from conversational data using Disco, I won't fully cover all its features. For that, you'll want to take a look at the more comprehensive manual that can be downloaded from bit.ly/2CHtfnc.

Now that I've introduced process mining, I want to delve into something I mentioned briefly earlier—event log data. But what exactly is event log data or, more usefully in this case, to what types of event log datasets can process mining be applied?

Process mining requires event log datasets with at least the following three fields present in the data:

1. Case ID, which is the ID of an object that goes through the different events.
2. Activity ID, which defines the type of events that can happen to an object.

Figure 1 **Conversation with ConversationId 3**

ConversionId	TimeStamp	Utterance
3	2018-04-02 14:03:02	Can you please give me quotes of mortgage rates
3	2018-04-02 14:03:05	I am interested in 15 years only for refinance
3	2018-04-02 14:03:08	Excellent, I would be interested to proceed
3	2018-04-02 14:04:12	My email is x@x.com
3	2018-04-02 14:04:15	Thank you, I will be waiting to hear back

3. Timestamp, which is the date and time of a specific event that happened to a given object.

Of course, there can be additional fields present in the dataset, as well, that can specify further information about the object. In what follows, I'll explain how to represent conversational data as event log data.

Getting Conversational Data Ready for Process Mining

As part of the overall approach, I'm going to represent conversations as processes. Each process case or instance is, therefore, a specific conversation, and the intents of the different utterances in each conversation then become the different events or states of the process. Later on, I'll show how this can be helpful in deriving insights from the conversations.

In order to make things more concrete, I'll use an example of conversational data from the financial technology space. In the download accompanying this article you'll find a CSV file containing 10 different simulated conversations of users with a chatbot that's providing assistance regarding mortgages. Bear in mind that this conversational data is limited and contains only the necessary fields to which I'll apply process mining. In addition, for simplicity, I've chosen to include only the user utterances, not the system responses. If you wanted, you could decide to include the system responses or any other data you think is related, such as information pertaining to the chat sessions, user data and so on.

The fields included in the CSV dataset are:

1. ConversationID, which identifies the conversation in a unique way and is mapped here to the Case ID.
2. Utterance, which is the user's utterance and is essentially unstructured text data to which LUIS is applied to identify intents.
3. TimeStamp, which is the timestamp for a given Conversation ID/Utterance pair. I'll use this field later as my timestamp for process mining.
4. Intent, which is an intent identified by LUIS. This field will be mapped as the Activity ID for process mining.

Note that while it's possible to use LUIS to identify both intents and entities (references to real-world objects that appear in an utterance), I've chosen, for simplicity again, to focus on intents only. The following intents are included in the data:

- GreetingIntent: a greeting or conversation opener.
- ExplorationIntent: a general exploratory utterance made by the user.
- OperatorRequestIntent: a request by the user to speak with a human operator.
- SpecificQuestionIntent: a question from the user about mortgage rates.
- ContactInfoIntent: contact information provided by the user.
- PositiveFeedbackIntent: positive feedback provided by the user.
- NegativeFeedbackIntent: negative feedback provided by the user.
- EndConversationIntent: ending of the conversation with the bot initiated by the user.

I then identified certain intents for each utterance in the conversation shown in **Figure 1** (in order): ExplorationIntent, SpecificQuestionIntent, PositiveFeedbackIntent, ContactInfoIntent and EndConversationIntent. In this way, the original conversational data is transformed into sequences of intents. The result of this transformation will be used as an input to Disco.

Applying Process Mining to Conversational Data Using Disco

To apply Disco, you'll have to import the CSV file with the conversational data and map the three required fields described earlier—ConversationID, TimeStamp and Utterance. This is shown in **Figure 2**.

After importing the CSV, Disco will give you access to three views that I'll go through one at a time: Map, Statistics and Cases.

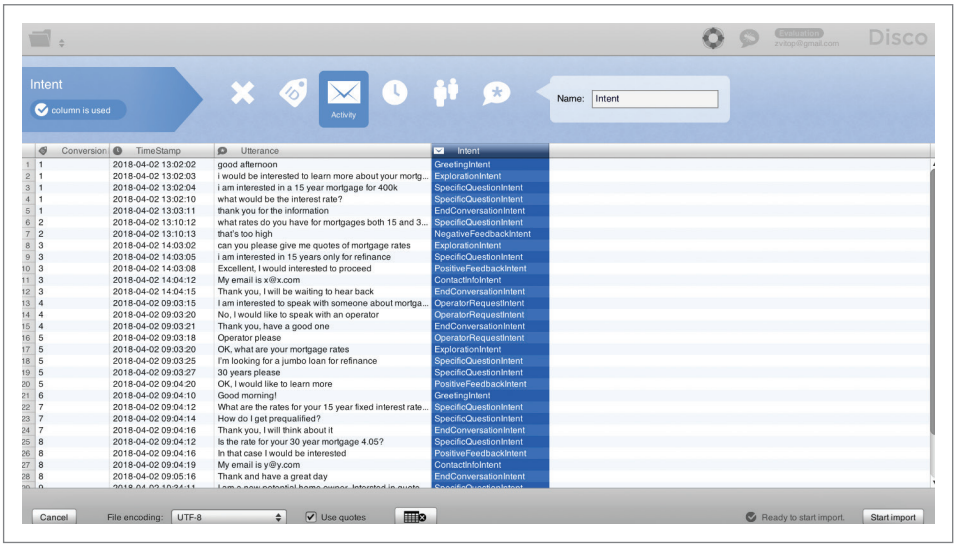
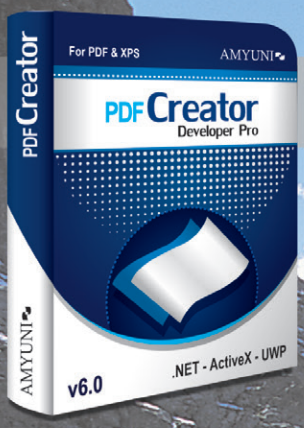


Figure 2 **Importing Conversational Data into Disco**

Switch to Amyuni® PDF

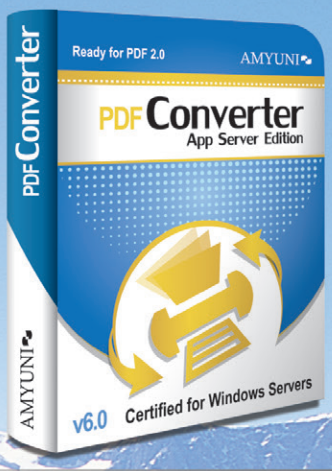
AMYUNI



Create and Process PDFs in .NET, COM/ActiveX and UWP

NEW v6.0

- Edit, process and print PDF 2.0 documents
- Create, fill-out and annotate PDF forms with Javascript support
- Fast and lightweight 64-bit components
- Universal Apps DLLs enable publishing C#, C++, CX or JS apps to Windows Store
- Plus a number of exciting features in v6.0



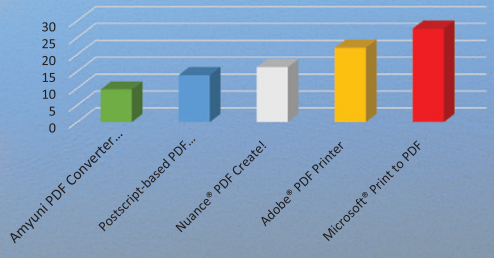
High Performance PDF Printer for Desktops and Servers

Print to PDF in a fraction of the time needed with other tools. WHQL certified for all Windows platforms. Version 6.0 updated for Server 2016.

Complete Suite of PDF, XPS and DOCX Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment
- Includes our Microsoft® WHQL certified PDF and DOCX printer driver
- Export PDF documents into other formats such as images, HTML5 or editable DOCX
- Import/Export XPS files using our native libraries

Benchmark Testing - Amyuni vs Others
Seconds required to convert a document to PDF



Other Developer Components from Amyuni®

- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract 3 engine for accurate character recognition
- Javascript Engine: Integrate a full JS interpreter into your PDF processing applications
- USB Mobile Monitor: Mirror the display of your Windows or Linux system onto your Android device

Windows Server 2016 Certified

Windows Server 2012 Certified

Windows

All development tools available at
www.amyuni.com

AMYUNI Technologies

USA and Canada
Toll Free: 1866 926 9864
Support: 514 868 9227
sales@amyuni.com

Europe
UK: 0800-015-4682
Germany: 0800-183-0923
France: 0800-911-248

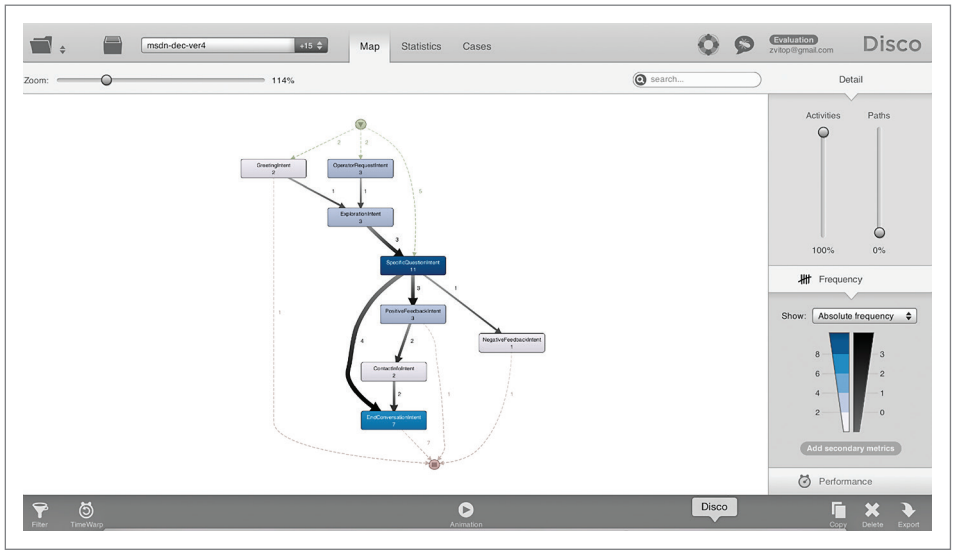


Figure 3 **Map View in Disco**

The first view, Map, is the map of the process discovered by Disco from the conversational data. The map is a graphical representation of the different transitions in the process between the events, as well as frequencies and repetitions of different events. In this case, those are the different transitions between the intents. **Figure 3** shows the map discovered by Disco for the conversations.

From **Figure 3**, you can get a general overview of the conversations and see that conversations can start in one of three different ways—a greeting, an operator request or a mortgage-specific question, with mortgage-specific questions being very frequent. Most conversations end with an EndConversationIntent, but a few end with other intents that represent greetings and negative feedback. In particular with regard to negative feedback, these can point to outlier conversations that may require more attention.

Moreover, transitions between different intents can also provide very useful information for deriving intents. For example, it may be possible to determine whether there are specific utterances or intents that lead to the intent representing negative feedback. It might then be desirable to drive conversations away from that path.

As mentioned earlier, information about repetitions of both intents and transitions is readily available as part of the Map view. In particular, you can see that the two most common intents in this case are SpecificQuestionIntent and EndConversationIntent, and that transitions from the former to the latter are very common. This provides a good summary at a glance regarding the content of the conversations.

It can also present an opportunity to improve conversations by considering breaking down SpecificQuestionIntent and EndConversationIntent into finer grain intents that can capture more insightful aspects of the user interaction. This should be followed by retraining LUIS and repeating the application of process mining to the modified conversational data.

Now let's take a look at the Statistics view, which is depicted in **Figure 4**. Here you can get insights about the duration of the conversations. Note that the conversations are now grouped into what Disco calls "Variants," which are essentially similar conversation flows that have the same intents and transitions between the intents. The Overview part of the Statistics view allows you to get summary statistics, such as median and mean, as well as information about end-to-end durations of different conversations. This can be useful to identify outliers, such as extremely short conversations, and to cross check with conversations from the Map view regarding potentially problematic conversations. It's also possible to identify conversations with longer durations. In the example I use here, those are likely to be successful conversations.

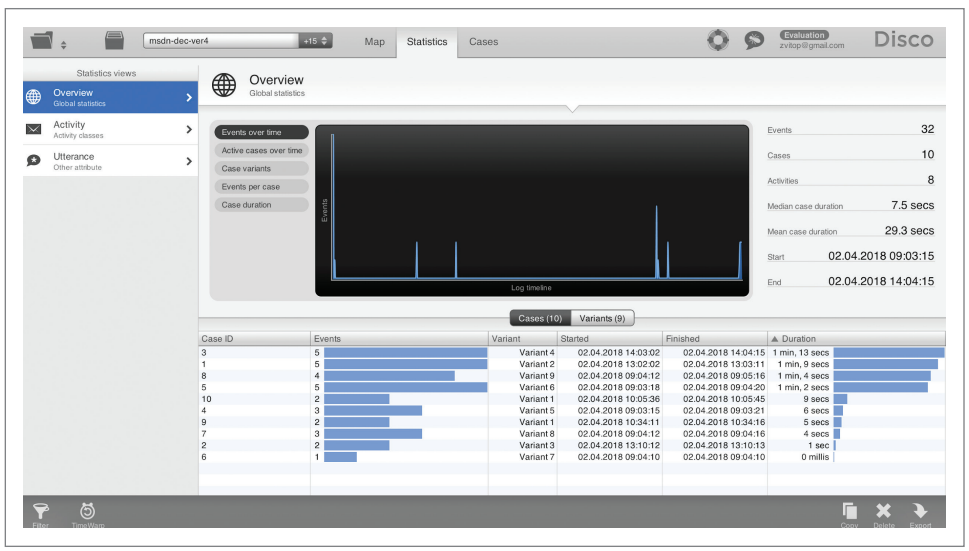


Figure 4 **Overview and Summary Statistics in Disco**

In order to dive deeper into conversations that exhibit interesting behaviors, for example, unusually long or short conversations or conversations with certain intent structures, you can use Disco's powerful filtering capabilities. At any given point, Disco allows you to filter the overall dataset by various dimensions. As an example, you can identify the conversation IDs that are of specific interest, apply the appropriate filter and then look at the different views in a filtered mode. This allows you to identify patterns common to the filtered conversations.

Disco also lets you look at the conversations at the intent level (by

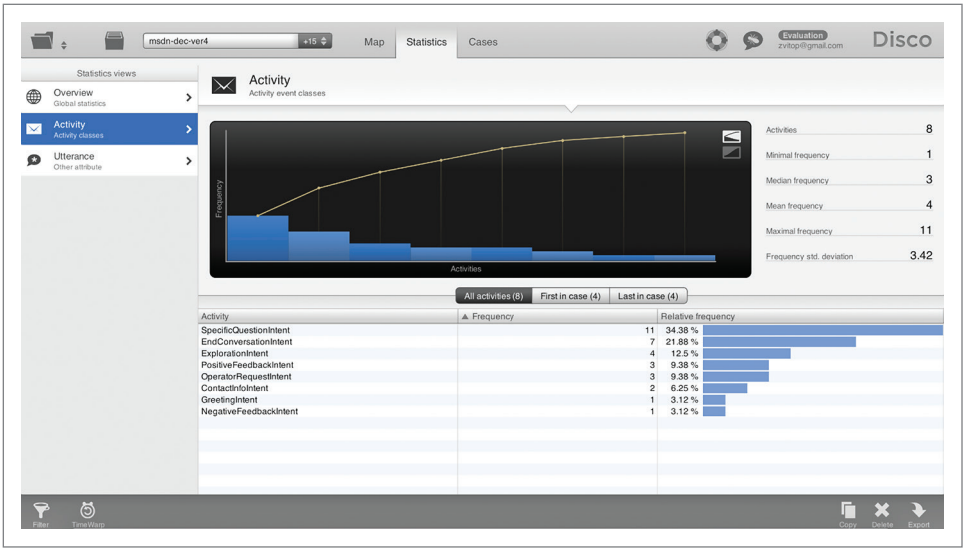


Figure 5 **Summary Statistics About Intents in Disco**

using the Activity section of the Statistics view), specifying different summary statistics pertaining to the intents. This is depicted in **Figure 5**. Here you get a distribution of the different intents in your conversations and can see that, fortunately, the negative feedback intent comprises only about 3 percent of the intents in your conversations.

The final analysis view to cover is the Cases view, which shows the different conversations based on their variant types. For example, in **Figure 6** you can see a specific conversation, with conversation ID 9, that belongs to a variant with two intents: SpecificQuestionIntent and EndConversationIntent. This view is very useful for comparing conversations having similar structures. It can help you to learn, for example, if there are any patterns you can adopt that would help make conversations more successful or, if you happen to find unexpected differences, it can help you discover what's causing them.

Before wrapping up, I want to note that the insights and the features of Disco I've reviewed so far are not comprehensive, but rather demonstrate the benefits of applying process-mining techniques to conversations modeled as processes.

As mentioned earlier, you can explore using many additional fields as part of your activity representation. You may want to include, for example, information about specific entities in user utterances; the responses of your conversational interface; or data about your users, such as locations, previous interactions with the system, and so on. Such rich representations will enable you to enhance the depth of insights from your conversational data and build better conversational systems. I strongly encourage you to explore more.

Wrapping Up

I've introduced process mining and have shown how to leverage that technology in conjunction with LUIS to derive insights from conversational data. In particular, LUIS is applied to the different utterances in the conversations to transform unstructured utterance text to structured intent labels. Then, through mapping of conversation ID, time stamps and intents to process-mining fields, and by using Disco, I showed how to apply process mining to the structured conversational data. Using the different views provided by Disco, from process discovery that shows overall conversation structure to grouping of conversations into different variants, it's possible to derive insights from the transformed conversational data, such as what makes conversation successful and how to use that knowledge to improve conversations that are less successful.

Keep in mind that this article just scratches the surface of what's possible when applying process mining to conversational data. I hope that using the resources presented in this article, along with others you may find along the way, will enable you to leverage process mining to create better, more compelling conversational interfaces. ■

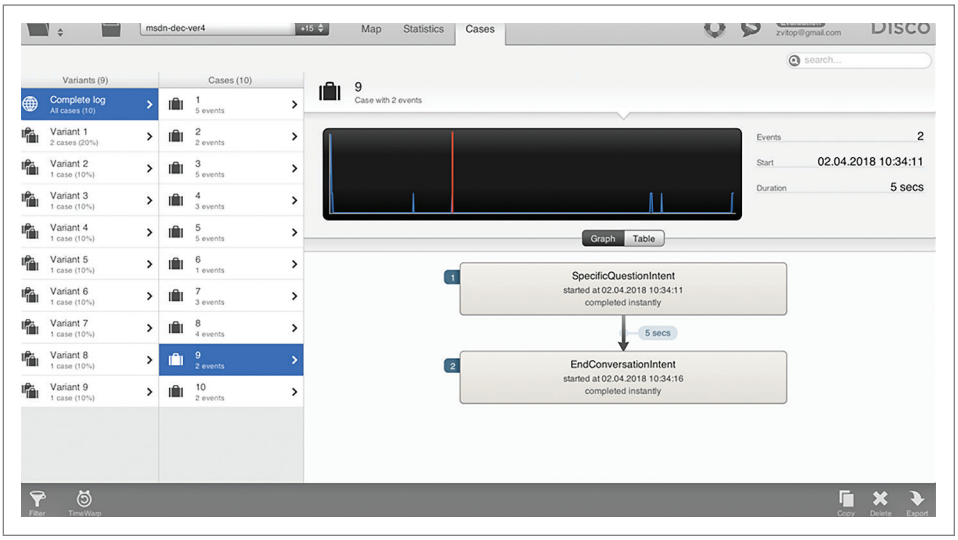


Figure 6 **The Cases View in Disco**

Zvi Topol *has been working as a data scientist in various industry verticals, including marketing analytics, media and entertainment, and Industrial Internet of Things. He has delivered and lead multiple machine learning and analytics projects including natural language and voice interfaces, cognitive search, video analysis, recommender systems and marketing decision support systems. Topol is currently with MuyVentive, an advanced analytics R&D company, and can be reached at zvi.topol@muyventive.com.*

Thanks to the following Microsoft technical expert who reviewed this article: *Sandeep Alur*

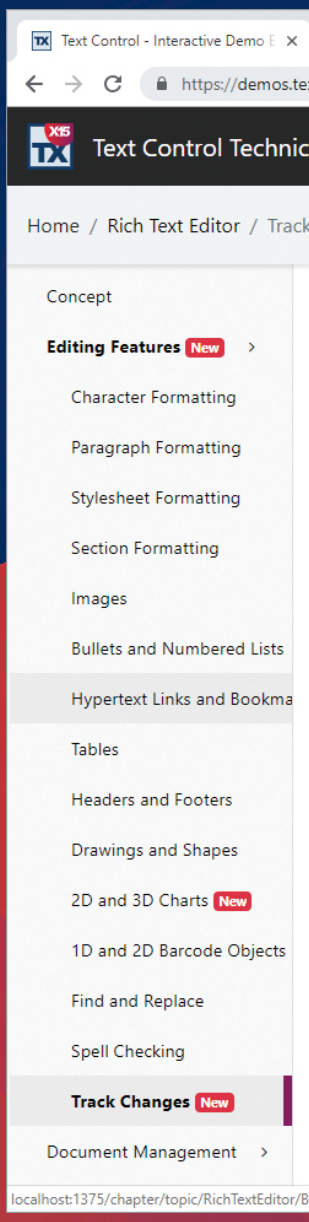
TEXT CONTROL

INTEGRATE DOCUMENT COLLABORATION

Integrate MS Word compatible track changes into cross-platform web applications. Share and review documents with a true WYSIWYG document editor.

See our technology live:

demos.textcontrol.com

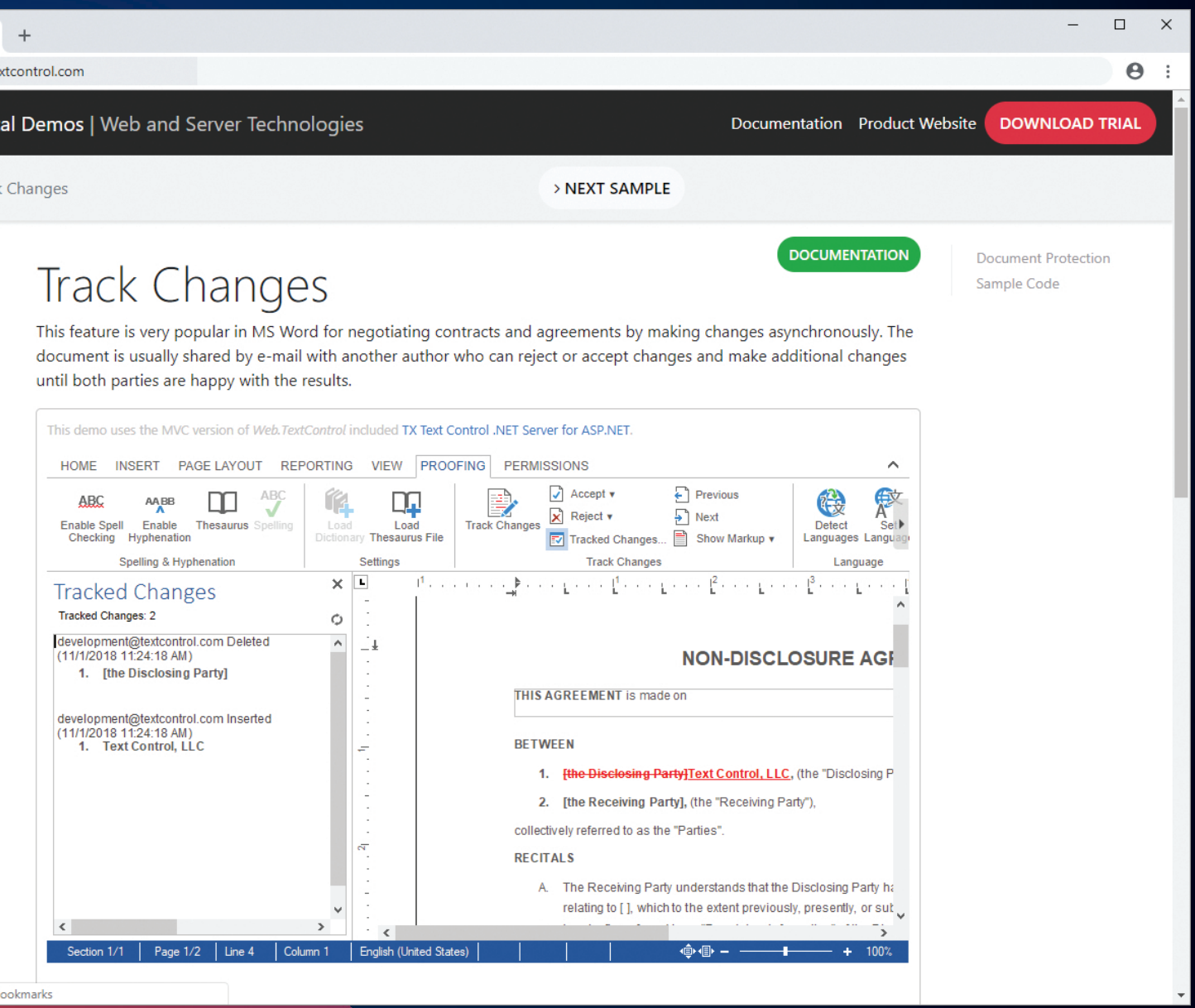


WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

Visual Studio
Partner

TX Text Control X16 Released

Evaluate our technology and test the most sophisticated cross-browser, and true WYSIWYG, rich text editor. Merge MS Word compatible templates with JSON data and create pixel-perfect Adobe PDF documents on-the-fly. See what's possible today!



Up and Running with Azure Kubernetes Services

Chander Dhall

In the world of container orchestration, everyone seems to being talking about Kubernetes. Originally designed by Google, Kubernetes is an open source platform for orchestrating Docker (or any Open Container Initiative) containers across clusters of virtual machines (VMs), with support for deployment, rollbacks, scaling and a host of other features. Administering a Kubernetes cluster can be a complex endeavor, so the Azure team has provided a managed solution called Azure Kubernetes Service (AKS) to make the process considerably easier.

In this article, I'll demonstrate how to deploy an AKS cluster, create a secure Azure Container Registry (ACR), deploy an ASP.NET Web API application, and expose that application on the Internet via a Kubernetes Ingress and Azure DNS. This is not intended to be an in-depth article on Kubernetes, but rather everything you need to get up and running quickly with an application using the Azure AKS offering.

One of the benefits of Azure AKS is that the control plane, consisting of the master and configuration nodes, is fully managed. The Kubernetes control plane typically comprises at least one master node and one, three or five etcd configuration nodes. As you can imagine, managing these core services can be tedious and costly. With AKS, you can upgrade the core services or scale out additional worker nodes with a click of a button. Additionally, at this time there are no additional charges for these management nodes. You only pay for the worker nodes that run your services.

The code discussed in this article can be found at bit.ly/2zaFQeq. Included in the repository is a basic ASP.NET Web API application, along with a Dockerfile and Kubernetes manifests.

This article discusses:

- Deploying an Azure Kubernetes Service (AKS) cluster
- Creating an Azure Container Registry (ACR) secure container registry
- Deploying and exposing an ASP.NET Web API application on the Internet via a Kubernetes Ingress and Azure DNS

Technologies discussed:

Azure Kubernetes Service, Azure Container Registry, Kubernetes Ingress, Azure DNS, ASP.NET Web API

Code download available at:

bit.ly/2zaFQeq

Creating a Kubernetes AKS Cluster

The Azure CLI is used for creating and managing resources in an Azure Cloud subscription. It can be found at bit.ly/20cwFQb. Throughout this article, I'll be using it to manage various Azure services. Azure Cloud Shell (bit.ly/2yESmTP) is a Web-based shell that allows developers to run commands without installing the CLI locally.

Let's get started by creating a resource group to hold the AKS cluster and container registry, with this bit of code:

```
> group create --name my-aks-cluster --location eastus2
```

Once the resource group is created, I create a cluster with a single node. While VMs as small as the B1 burstable images are allowed, it's suggested that you use at least a 2-core, 7GB memory instance (D-series or greater). Anything smaller has a tendency to be unreliable when scaling and upgrading a cluster. You'll also need to take into consideration that AKS currently only supports nodes of a single type, so you cannot decide to scale up to a larger VM instance at a later time. Adding nodes of a different type will be supported in the future with multiple node pools, but for now you need to choose a node size that fits the needs of the services that you plan to run.

> One of the benefits of Azure AKS is that the control plane, consisting of the master and configuration nodes, is fully managed.

Sit back and be patient while the cluster is being created, as it often takes upward of 10 to 12 minutes. Here's the code to kick off the operation:

```
> az aks create --resource-group my-aks-cluster --name my-aks-cluster
  --node-count 1 --generate-ssh-keys --kubernetes-version 1.11.2
  --node-vm-size Standard_D2s_v3
```

Getting Docker images into the AKS cluster requires the user of a Docker registry. Using a public Docker registry is acceptable for open source distributions, but most projects will want to secure application code in a private registry.

Azure provides a secure, managed Docker registry solution with Azure Container Registry (ACR). To setup an ACR instance, run the following command:

```
> az acr create --resource-group my-aks-cluster
  --name <REGISTRY_NAME> --sku Basic --admin-enabled true
```

Note that the registry name must be unique across all of the ACR-hosted registry names on Azure.

The Kubernetes CLI

Kubectl is used to interact with an AKS cluster. It's available for all OSes and has multiple approaches to installation. You can find more information at bit.ly/2Q58CnJ. There's also a Web-based dashboard that can be very helpful for getting a quick overview of the state of the cluster, but it doesn't cover every API operation available and you may often find yourself falling back to the kubectl CLI. Even if you're not a fan of command-line operations, over time you'll likely grow to appreciate the power of kubectl. In combination with the Azure CLI, any operation can be performed without leaving the shell.

Once kubectl has been installed, credentials can be imported locally to authenticate to the cluster using the following command:

```
> az aks get-credentials --resource-group my-aks-cluster
  --name my-aks-cluster
```

Running this command updates ~/.kube/config with your cluster uri and signing authority and credentials. It also adds a context for setting the cluster as the current configuration. The kubectl configuration can hold context for multiple clusters, which can easily be switched using the kubectl config command. Additionally, there are open source utilities available to make switching contexts easier (kubectx and kubectxwin).

Once the credentials have been imported, connectivity to the cluster can be tested by listing the running nodes with the kubectl get nodes command. You should see something like this:

```
> kubectl get nodes
NAME                       STATUS    ROLES     AGE       VERSION
aks-default-34827916-0     Ready     agent     1d        v1.11.2
```

Adding a Container Registry Secret for Deployments

Kubernetes has a secure way to store sensitive data using Secrets. For instance, to prevent ACR credentials from being stored in the source code, a secret should be created and referenced from the Kubernetes deployment manifests. To retrieve the credentials for the ACR service, run the following command:

```
> az acr credential show --name <REGISTRY_NAME>
```

Next, use kubectl to generate a special type of secret (docker-registry) that's designed specifically to store credential tokens provided by Docker. The code that follows will create a secret called my-docker-creds using the credentials that were retrieved from the ACR query. Be aware that the username is case-sensitive and ACR will make it lowercase by default for the built-in admin account:

```
> kubectl create secret docker-registry my-docker-creds
  --docker-server=<REGISTRY_NAME>.azurecr.io --docker-username=<REGISTRY_USERNAMEE>
  --docker-password=<REGISTRY_PASSWORD> --docker-email=<ANY_EMAIL>
```

> All applications in Azure Kubernetes Service are deployed as Docker containers.

Finally, confirm that the secret was created by running the following command:

```
> kubectl describe secrets my-docker-creds
Name:         my-docker-creds
Namespace:    default
Type:  kubernetes.io/dockerconfigjson
Data
====
.dockerconfigjson:  223 bytes
```

Creating a Docker Container

All applications in AKS are deployed as Docker containers. Here's the code for a Dockerfile that creates a Docker image that can be shipped to the cluster:

```
FROM microsoft/dotnet:2.1-sdk AS builder
COPY . /app
WORKDIR /app
RUN dotnet publish -f netcoreapp2.1 -c Debug -o /publish

FROM microsoft/dotnet:2.1.3-aspnetcore-runtime
COPY --from=builder /publish .

ENTRYPOINT ["/bin/bash", "-c", "dotnet WebApiApp.dll"]
```

This Dockerfile uses a multi-stage approach that splits the build into separate stages for build and runtime. This reduces the size of the overall image significantly by not including the entire SDK for distribution.

Pushing the Image to the Registry

Docker works on the concept of local images and containers to execute images. A Docker image cannot be pushed directly to the cluster. Instead, the image must be hosted in a location that Kubernetes can access to pull the image locally to the cluster. The ACR registry is a secure location that allows images to be managed centrally between development, continuous integration and cluster environments.

Docker works on the concept of local images and containers to execute images. A Docker image cannot be pushed directly to the cluster.

The image must be built and tagged with the format <REGISTRY>/<REPOSITORY>/<IMAGE>:<TAG> so that Docker will know where to push the image upstream. The repository, which can have any name, provides a way to separate out registry images into logical groups. The following code demonstrates how to build and tag an image before pushing to ACR. While the *latest* tag is supported, when working with Kubernetes it's highly advisable to use semantic versioning. It makes managing deployments and rollbacks much easier when you can leverage version numbers. Here's the code:

```
> az acr login --name <REGISTRY_NAME>
> docker build -t <REGISTRY_NAME>.azurecr.io/api/my-webapi-app:1.0 .
> docker push <REGISTRY_NAME>.azurecr.io/api/my-webapi-app:1.0
baf6b1178a5b: Pushed
b3f8eefa2758: Pushed
393dd8f4a879: Pushed
0ad9ffac9ae9: Pushed
8ea427f58308: Pushed
cdb3f9544e4c: Pushed
1.0: digest: sha256:47399f3b2365a9 size: 1579
```

Now confirm that the image was pushed upstream by running:

```
> az acr login --name <REGISTRY_NAME>
> az acr repository list --name <REGISTRY_NAME>
```

Figure 1 **Deployment.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-webapi-app
spec:
  selector:
    matchLabels:
      app: my-webapi-app
  replicas: 2
  template:
    metadata:
      labels:
        app: my-webapi-app
    spec:
      containers:
      - name: my-webapi-app
        image: <REGISTRY_NAME>.azurecr.io/api/my-webapi-app:1.0
        livenessProbe:
          initialDelaySeconds: 10
          path: /health
          periodSeconds: 5
        ports:
        - containerPort: 80
      imagePullSecrets:
      - name: my-docker-creds

Deploy the manifest with the following command:
> kubectl apply -f ./deployment.yaml
```

Deploying the Application

Kubernetes uses manifests to describe every object in the cluster. Manifests are yaml files that are managed through the Kubernetes API. A deployment manifest type is used to describe the resources, image source and desired state of the application. **Figure 1** is a simple manifest that tells Kubernetes which container to use, the number of desired running instances of the container and labels to help describe the application in the cluster. It also adds the name of the secret to authenticate to ACR when pulling the remote image.

Kubernetes uses a concept called a pod to group one or more containers into a logical, scalable instance within the cluster. Typically, you'll have one container per pod. This allows you to independently scale any service for your application. A common misconception is to put all the services of an application—such as Web Application and Database—in a single pod. Doing this doesn't allow the Web front end to scale individually from the database, and you'll lose many of the benefits of Kubernetes as a result.

There's a common scenario where it's acceptable to have an additional container in a pod—it's a concept called a sidecar. Imagine a container that observes your application container and provides metrics or logging. Placing both containers in a single pod provides real benefits in this instance. Otherwise, it's generally best to keep the ratio of one container per pod until you have a solid understanding of the limitations (and benefits) of grouping containers.

Once the deployment has completed, the status of the application pod can be checked with the following command:

```
> kubectl get pods
NAME                              READY     STATUS    RESTARTS   AGE
my-webapi-app-64cdf6b449-9hsks    2/2       Running   0          2m
```

Note that two instances of the pods are running to satisfy the replica set.

Creating a Service

Now that the application Docker container is deployed into the cluster, a Service is required to make it discoverable. A Kubernetes Service makes your pods discoverable to other pods within the cluster. It does this by registering itself with the cluster's internal DNS. It also provides load balancing across all of the pod replicas, and manages pod availability during pod upgrades. A service is a very powerful Kubernetes concept that's required to provide availability during rolling, blue/green, and canary deployment upgrades to an application. The following command will create a service for the deployment:

```
> kubectl expose deployment/my-webapi-app
service "my-webapi-app" exposed
```

Figure 2 **Confirm Running Routing Services**

```
> kubectl get pods --all-namespaces
NAMESPACE     NAME                                                        READY
kube-system   addon-http-application-routing-default-http-backend-74d455htfw9  1/1
kube-system   addon-http-application-routing-external-dns-7cf57b9cc7-lqhl5    1/1
kube-system   addon-http-application-routing-nginx-ingress-controller-5595b2v  1/1
```

Now run the following command to view the service running in the cluster:

```
> kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
my-web-api   ClusterIP   10.0.0.157   <none>        443/TCP   1d
```

By default, services are only accessible from within the cluster, hence the absence of an external-ip. The kubectl CLI provides a convenience to open a proxy between the local machine and the cluster to interactively check if it's running, which can be seen in this code:

```
> kubectl port-forward services/my-webapi-app 8080:80
> curl http://localhost:8080
StatusCode        : 200
StatusDescription : OK
Content           : Hello from our Kubernetes hosted service!
```

> Kubernetes is secure by default and you must explicitly expose services that you wish to access from outside the cluster.

Adding HTTP Routing

Kubernetes is secure by default and you must explicitly expose services that you wish to access from outside the cluster. This is an excellent design feature from a security perspective, but can be confusing to a first-time user. The most common way to access HTTP-based services inside the cluster is by using a Kubernetes Ingress controller. Ingress controllers provide a way to route requests to internal services based on a hostname and path through an HTTP proxy entrypoint.

Before Ingress was added to Kubernetes, the primary way to expose a service was by using a LoadBalancer service type. This would cause a proliferation of load balancers—one per service—that would each need to be separately managed. With Ingress, every service in the cluster can be accessed by a single Azure Load Balancer, significantly reducing cost and complexity.

Figure 3 **Ingress.yaml**

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-webapi-app
  annotations:
    kubernetes.io/ingress.class: addon-http-application-routing
spec:
  rules:
  - host: my-webapi-app.<CLUSTER_PREFIX>.eastus2.aksapp.io
    http:
      paths:
      - path: /
        backend:
          serviceName: my-webapi-app
          servicePort: 80
```

AKS provides a convenient add-on to extend the cluster with an Nginx proxy that acts as an Ingress controller for handling these requests. It can be enabled via the Azure CLI with the following command:

```
> az aks enable-addons --resource-group my-aks-cluster
  --name my-aks-cluster --addons http_application_routing
```

You can confirm that the routing services are running by issuing the command shown in **Figure 2**.

You should see three new controllers in the list. The Ingress controller, external DNS and the default back end. The default back end is used to provide a response to clients when a route can't be found to an existing service. It's very similar to a 404 Not Found handler in a typical ASP.NET application, except that it runs as a separate Docker container. It's worth noting that while HTTP add-on is great for experiments, it's not intended for production use.

Exposing the Service

An Ingress is a combination of an Ingress controller and an Ingress definition. Each service will have an Ingress definition that tells the Ingress controller how to expose the service. The following command will get the DNS name for the cluster:

```
> az aks show --resource-group my-aks-cluster
  --name my-aks-cluster --query
  addonProfiles.httpApplicationRouting.config.HTTPApplicationRoutingZoneName
  -o table
Result
------------------------------------
<CLUSTER_PREFIX>.eastus2.aksapp.io
```

The ingress annotation *kubernetes.io/ingress.class* notifies the Ingress controller to handle this specification, as shown in **Figure 3**. Using the cluster DNS name resolved earlier, a subdomain will be added for the host along with a "/" root path. Additionally, the service name and its internally exposed port must be added to tell the Ingress controller where to route the request within the cluster.

The Ingress manifest can then be applied with this command:

```
> kubectl apply -f ./ingress.yaml
```

It can take a few minutes for the DNS entries to be created and propagated, so please be patient. The status of the DNS service can be checked from within the Azure Portal, like so:

```
> curl http://my-webapi-app.<CLUSTER_PREFIX>.eastus2.aksapp.io
StatusCode        : 200
StatusDescription : OK
Content           : Hello from our Kubernetes hosted service!
```

Wrapping Up

At this point, we have a single-node AKS cluster running alongside an ACR service hosting the application Docker images with secured secrets. This should be a great starting point for exploring the many additional capabilities that Azure AKS Kubernetes can offer. I have a simple philosophy, "Buy what enables you and build what differentiates you." As you can see, Azure simplifies Kubernetes so that both developers and DevOps professionals can focus on more critical tasks. ■

Chander Dhall, *CEO of Cazton, is an eight-time awarded Microsoft MVP, Google developer expert and world-renowned technology leader in architecting and implementing solutions.*

Thanks to Microsoft for providing a technical review of this article: *Brendan Burns*

Rapid IoT Development with Azure IoT Central

Dawid Borycki

As an IoT developer, you face many challenges. Luckily, Microsoft IoT technologies have you covered. You can use Windows 10 IoT Core to develop Universal Windows Platform (UWP) apps for smart devices (bit.ly/2yJf6RJ). You can create machine learning (ML) algorithms graphically with Azure Machine Learning Studio (bit.ly/2yF2yes). And you can choose among various approaches to create Web apps using many Azure IoT services or self-developed ASP.NET MVC apps (either .NET Framework or .NET Core).

Though these technologies offer a comprehensive way to develop custom IoT solutions, combining them can be difficult, especially if you don't have prior cloud or Web programming experience. To solve this problem, Microsoft created Azure IoT Suite (see my previous article at bit.ly/2yFaIU6), which was later renamed to Azure IoT solution accelerators (bit.ly/2pYaraX). Solution accelerators provide a number of preconfigured apps that target typical IoT problems, including dashboards, ML models, streaming data logic and programmatic components that tie everything together. However, solution accelerators are still complex. So, to simplify IoT development even further, Microsoft introduced Azure IoT Central, a cloud-based managed service you can use to quickly create an IoT back end. This is the modern portal, which contains dashboards and underlying services for telemetry, data processing and more. You use this back end to connect, monitor and manage your IoT devices.

In this article, I'll show how to use Azure IoT Central to create the solution shown in **Figure 1**. This solution uses a custom IoT Central app that depicts telemetry data, device location and its settings, and two key performance indicators. An operator can use this dashboard to visualize the telemetry and remotely control the device through settings. The telemetry data is streamed from the .NET Core console application (shown in the top right corner). The full source code of this app is available at bit.ly/2D34XnV.

This article discusses:

- Approaches to developing IoT solutions with Azure
- How to create an Azure IoT Central application for remote monitoring
- How to connect the .NET Core client app to Azure IoT Central

Technologies discussed:

Azure IoT, .NET Core, C#

Code download available at:

bit.ly/2D34XnV

Azure IoT Services and Solutions

Before I explain how I created the IoT solution, I'll briefly review other possible approaches to developing IoT applications with Azure IoT. First, you can create a fully custom solution by instantiating and manually configuring dedicated Azure IoT services, as shown in **Figure 2**. In this case, you typically start with the IoT Hub, which acts as the cloud gateway and is used for bi-directional communication and device management (for example, device

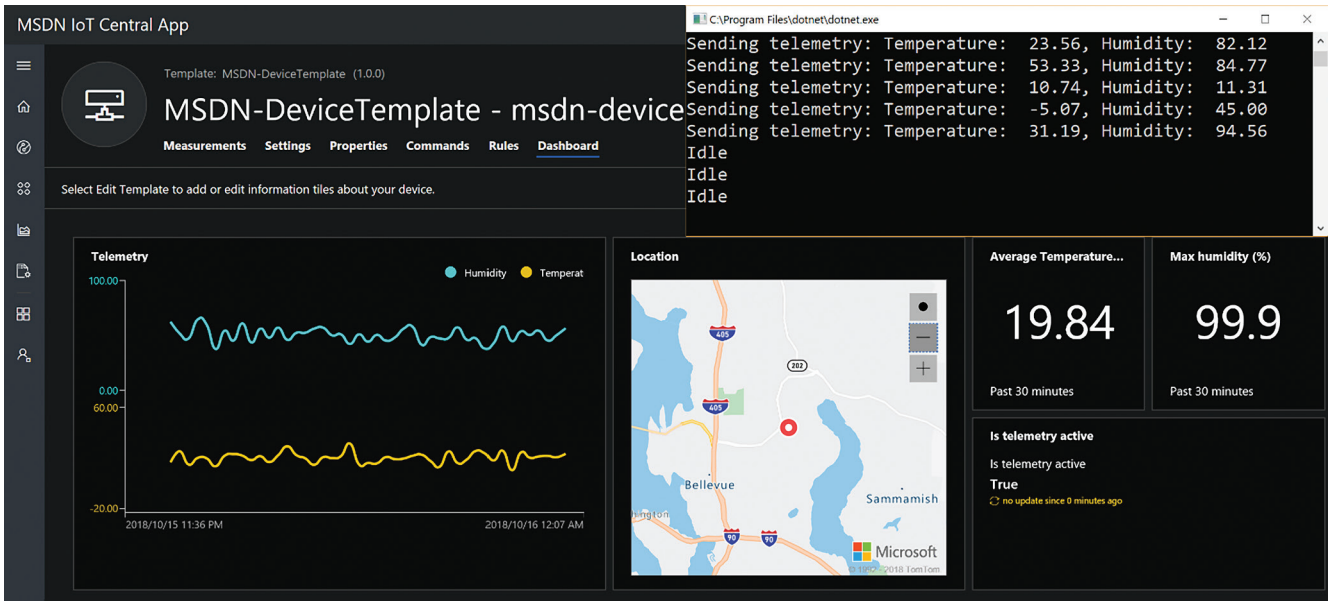


Figure 1 **The IoT Solution I'll Create**

registration). Data transmitted from the remote devices through the IoT Hub can then be preprocessed or transformed with Azure Stream Analytics. You use this service to filter out non-telemetry data or to average short portions of the telemetry data to reduce rapid fluctuations in your measurements. The preprocessed data can then be sent to the Power BI dashboard for visualization, persisted in dedicated cloud storage or transmitted to Event Hub for more complex analysis. Event Hub can send preprocessed data to an ML model to detect anomalies or predict trends in your monitored process. However, such an approach can be time-consuming, mostly because you don't have a preconfigured Web portal to provide a convenient interface for your users and operators.

Second, you can start with IoT solution accelerators, which are templates you can adjust to your needs. IoT solution accelerators are based on the same Azure IoT services as in a fully custom solution. However, accelerators come with preconfigured services and sample code that show you how to take advantage of those services. In this case, you still have some work to do, but you start with adjustable templates.

Third, you can rapidly create a back end with Azure IoT Central, in which you're given a fully prepared IoT application. The amount of work and necessary skills are minimized. You don't need any cloud knowledge to create a fully functional, scalable and modern IoT solution. Everything is created for you automatically based on the same Azure IoT services as in a fully custom solution or when using solution accelerators. Hence, you can focus on developing your own things without taking care of other parts of your IoT solution. In what follows I'll use Azure IoT Central to rapidly create the cloud endpoint.

Creating an IoT Central Application

To create the IoT Central app I used the IoT Central portal at apps.azureiotcentral.com, which displayed the Application Manager when I logged in with my Microsoft account. This manager shows all your apps, though initially, of course, there won't be any apps. So, I clicked the New Application button, opening the IoT application creator, which lets you choose either a seven-day free trial or a paid subscription (bit.ly/2QLvk4t), as well as the application template, application name and the URL. I chose the Free plan and the Custom Application template, then set the app name to MSDN IoT Central app, which automatically created the following URL: msdn-iot-central-app.azureiotcentral.com. Finally, I clicked the Create button and, after several seconds, the app was ready.

The IoT Central application consists of two important elements: the actual view and the navigation (the left side bar). You use the side bar to switch among the various views: Home, Device Explorer, Device Sets, Analytics, Jobs, Application Builder and Administration. In this article I'll mostly work with the Device Explorer and Application Builder views.

The newly created app shows the default form of the homepage. You can customize this view by clicking the blue Edit button to activate edit view, in which you can add several components such as links, labels, images, device settings, maps and so on. However, before creating a dashboard you need telemetry data. Initially,

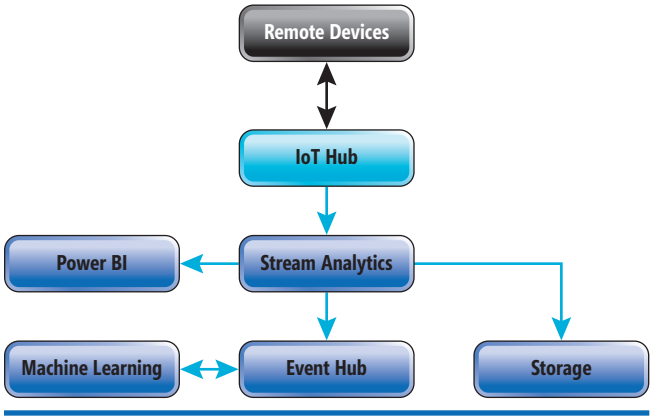


Figure 2 **A Typical IoT Solution: Rectangles Denote Services and Arrows Show the Data Flow Between Solution Components**

telemetry will be produced by the simulated device. To add such a device, you can start by clicking the Create Device Template panel on the default Home Page. Eventually you may prefer to use the Application Builder, which lets you choose the Custom Device option. The device template defines your device, including telemetry data, settings, events and remote commands (bit.ly/2CjsoYH).

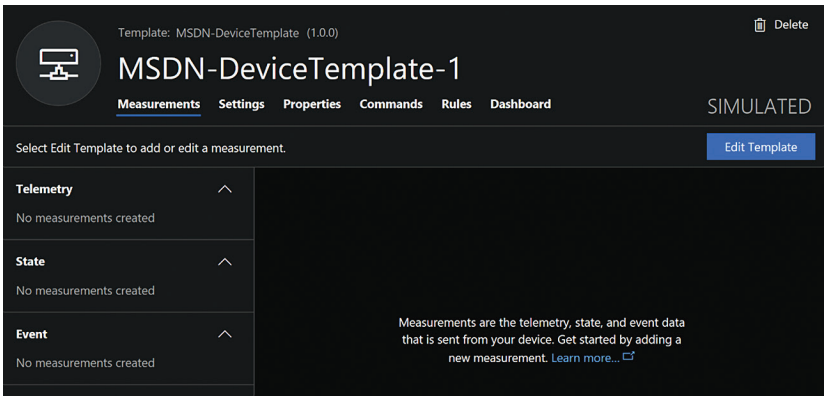


Figure 3 **A Device Template**

Device Template

No matter how you choose to create the device template, you first specify the template name (here, I'm using MSDN-DeviceTemplate), then click the Create button and the device will be provisioned. The view shown in **Figure 3** appears. Use this view to configure various aspects of your device:

- **Measurements:** Lets you specify what kind of data your device will provide to the cloud. You can also define device states and events.
- **Settings:** Lets you create device-specific settings that can be used to parameterize each device.
- **Properties:** Lets you configure device properties, for example its physical location.
- **Commands:** Lets you define commands, which can be sent from the cloud to the device in order to update its state.
- **Rules:** Lets you define rules for your device. These rules can monitor your data and trigger corresponding actions.
- **Dashboard:** Lets you create the device's dashboard. You use such a dashboard to create the summary for your device, which can include telemetry plots, images, key performance indicators, a map indicating device location, state and event history.

Telemetry, Settings and Properties

Next, I defined the device template. I created two measurements: temperature and humidity, then I defined a setting—IsTelemetryActive—that allows the operator to remotely enable or disable the telemetry (the Idle state in **Figure 1**). If the IsTelemetryActive setting is false, the remote device won't stream any data. Last, the device will have one property, Location, which contains the geospatial address of the device.

To create telemetry measurements, you first click the Edit Template button. As **Figure 4** shows, the New Measurement button will appear right above the Telemetry. After clicking this button you're given an option to choose between Telemetry, State and Event. I'll click Telemetry, which activates another view, which you use to configure temperature measurement in the following way (see middle column of **Figure 4**):

- **Display Name:** Temperature
- **Field Name:** Temperature
- **Units:** degC
- **Minimum Value:** -20
- **Maximum Places:** 2
- **Color:** choose whichever you like

Most of the telemetry settings are self-explanatory, but keep in mind that the Field Name is the name that's used while transmitting your data between your device and the cloud. So you use Field Name to properly serialize and deserialize your objects, representing telemetry data.

Defining a Humidity measure is much the same as Temperature, but use the following configuration:

- **Display Name:** Humidity
- **Field Name:** Humidity
- **Units:** %
- **Minimum Value:** 0
- **Maximum Value:** 100
- **Decimal Places:** 2

After preparing measurements, you can add the device setting. To do so, click the Settings tab and ensure you're in edit mode. You'll then see the available setting types on the right. Choose the Toggle type and configure it as follows:

- **Display Name:** Is telemetry active
- **Field Name:** IsTelemetryActive
- **ON Display Text:** True
- **OFF Display Text:** False

Finally, you add the device property: Location. To do so, open the Properties tab. The list of available property types will appear

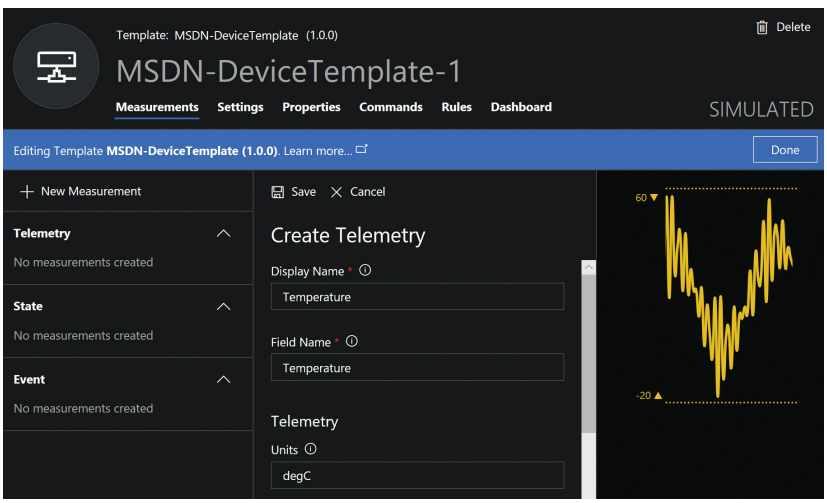


Figure 4 **Configuring Telemetry**

Manipulating Documents?

APIs to view, convert, annotate, compare, sign, assemble and search documents in your applications.

Try GroupDocs APIs for FREE

Download a Free Trial at

https://downloads.groupdocs.com

GROUPDOCS

GroupDocs.Viewer

View over 50 documents and image formats in any application using document viewer APIs.

GroupDocs.Annotation

Add annotations to specific words, phrases and any region of the document.

GroupDocs.Conversion

Fast batch document conversion APIs for any .NET, Java or Cloud app.

GroupDocs.Comparison

Compare two documents and get a difference summary report.

GroupDocs.Signature

Digitally sign Microsoft Word, Excel, PowerPoint and PDF documents.

GroupDocs.Assembly

Document automation APIs to create reports from templates and various data sources.

GroupDocs.Metadata

Organize documents with metadata within any cross platform application.

GroupDocs.Search

Transform your document search process for advance full text search capability.

- GroupDocs.Text
- GroupDocs.Editor
- GroupDocs.Parser
- GroupDocs.Watermark

Americas: +1 903 306 1676 EMEA: +44 141 628 8900 Oceania: +61 2 8006 6987
sales@asposeptyltd.com

Figure 5 **Configuring the Chart**

on the left. This list is referred to as a library. Click the Location object. Then, configure the property as follows:

- **Display Name:** Device Location
- **Field Name:** Location
- **Initial Value:** Microsoft, 1 Microsoft Way, Redmond, WA 98052

Note that the Initial Value field includes an auto-suggestion list, which is populated when you start typing.

After configuring the telemetry, settings and properties, I can now prepare the device dashboard, which will combine all the information in a modern graphical interface.

Creating a Dashboard

Creating the dashboard is similar to creating Settings and Properties—you first enable the template edit mode, which will open a library containing the list of available UI components. The components are located on the left side of the Azure IoT Central dashboard. When you choose an item, it will appear in the dashboard. The left side of the template editor will then display configurable settings for the UI component, as shown in **Figure 5**. Go ahead and click Line Chart. Next, set the Title to Telemetry, enable all three toggles, set the time range to Past 30 minutes, and then click the rightmost icon next to Humidity and Temperature so both these measures will be plotted.

I then added a Map to the dashboard, showing the device location. To create such a map, you choose the Map UI component from the library, then configure its properties. Set the Title to Location and, from the Location Property list, choose Device Location.

Next, I created two tiles showing key performance indicators (KPIs). These KPIs are the average temperature and maximum humidity (see the right part of **Figure 1**), calculated from sensor readings obtained in the past 30 minutes. To create such tiles, you use KPI components from the library, configuring them as follows:

Average Temperature KPI:

- **Title:** Average Temperature
- **Time Range:** Past 30 minutes
- **Measurement Type:** Telemetry
- **Measure:** Temperature

Maximum Humidity KPI:

- **Title:** Max Humidity
- **Time Range:** Past 30 minutes
- **Measurement Type:** Telemetry
- **Measure:** Humidity

Finally, I created a tile showing the actual value of the IsTelemetryActive setting. To do this, I used the Settings and Properties UI component, which has two configurable options: Title (a text box) and Settings and Properties (a two-panel control showing available and selected columns). Use the text box to set the title to Device Location, then use the second control to drag the Device Location from the available to the selected columns. Once all UI components are in place I can position them within the dashboard as previously shown in **Figure 1**.

Provisioning a Real Device

The preceding discussion confirms that with IoT Central you can quickly create a modern-looking dashboard for your IoT solution. However, so far we've relied only on the simulated device. Let's see now how to connect the real device to the IoT Central app. You do this with the Device Explorer, proceeding as explained in IoT Central documentation (bit.ly/2Ch4gWA). Briefly, you first click the New button (located in the top panel), choose Real from the drop-down list, then provide the name and unique identifier of your device. Here, I set these two options to MSDN-DeviceTemplate – msdn-device-id1 and msdn-device-id-1, respectively (see **Figure 6**).

If you now click the real device in Explorer, you'll see that its telemetry, settings, properties and dashboard are all the same as that of the simulated device. However, you don't have any measurement yet because the device isn't connected. Also, in the top right corner of the real device template you'll see additional hyperlinks: Block

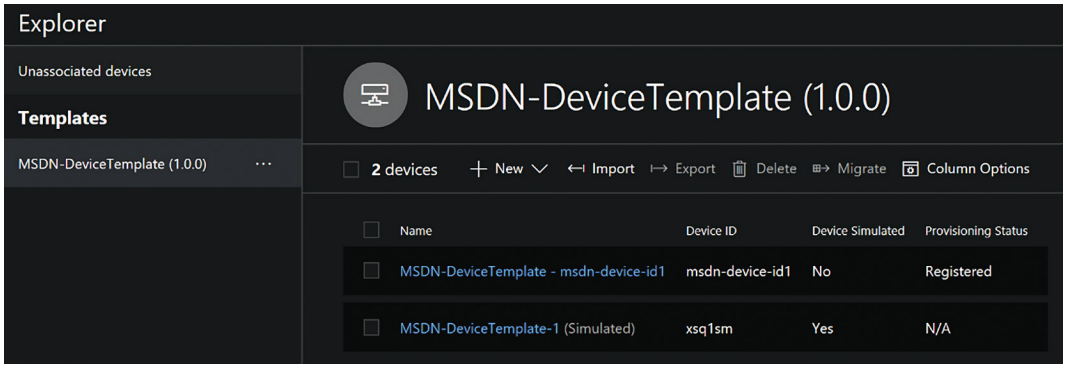


Figure 6 **Device Explorer Showing the Simulated and Real Devices**

and Connect. Block lets you block the device so the IoT Central app won't accept any requests from the remote device. Connect displays the Scope ID and credentials necessary to connect the device to the cloud, as shown in **Figure 7**.

IoT Central supports two ways to authorize your device. You can use either Shared Access Signature (SAS) or X.509 Certificates. Both approaches are described in detail at bit.ly/2ClDv3z. In the client app I'm going to develop, I'll use the SAS approach. So, to proceed further I'll need to note the Scope ID, Device ID and either the primary or secondary key displayed on the device connection screen in **Figure 7**. I'll use the dps_cstr command-line tool to generate the connection string. (You can download the Windows version of this tool from bit.ly/2Cj3Ejv.) Then, to obtain the actual connection string, you open the command line and type the following command:

```
dps_cstr <scope_id> <device_id> <SAS Key>
```

For the parameters shown in **Figure 7**, the dps_cstr tool generated the following connection string:

```
HostName=saas-iothub-28681fd2-94c7-4938-bf7e-7ae3e94a407c.azure-devices.
net;DeviceId=msdn-device-id1;SharedAccessKey=nQqFzf6TvnQA+zFI4MVaSSBeZgsY
SYOP7KXrl6z6oDE=
```

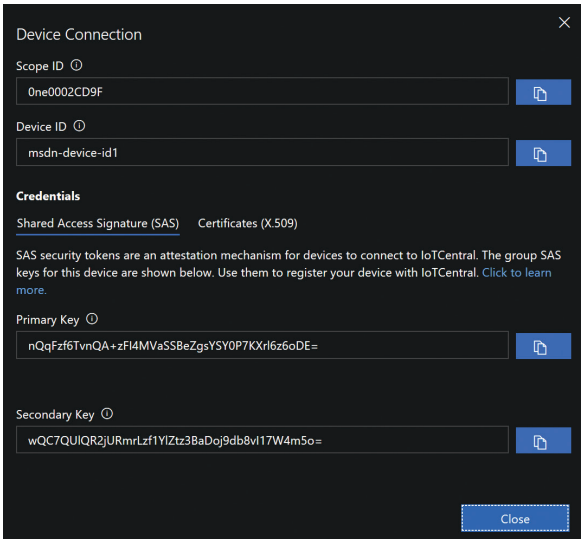


Figure 7 **Device Connection**

Implementing the Client App

To create the client app, I developed a C# .NET Core app based on the Console Application template. Most of the code could be used without any changes in a UWP Windows 10 IoT Core app, as well. However, I decided to use .NET Core in order to minimize the workloads required to implement the client app for IoT Central.

I used Visual Studio 2017 Community Edition, starting with the New Project dialog. I chose the Console App project template (.NET Core 2.1), then set the app name to IoTCentralClient, and installed the Microsoft.Azure.Devices.Client NuGet package to quickly connect to the IoT Hub.

I then proceeded to the actual implementation, starting with the DeviceClientHelper class shown in **Figure 8**.

The DeviceClientHelper class uses the Microsoft.Azure.Devices.Client.DeviceClient to associate the connection with the IoT Hub. To that end, DeviceClient uses the connection string generated by the dps_cstr tool. The connection string is passed as an argument to the CreateFromConnectionString static method of the DeviceClient class.

Telemetry

After preparing the connection, I created the Data class, shown in **Figure 9**, which is an abstract representation of the telemetry that will be sent to the cloud.

Remember that public properties of the Data class are used in the cloud endpoint to properly deserialize objects sent from your device. Hence, property names have to correspond to the Field Names used in the device template. Otherwise, your telemetry data will not be properly parsed by the cloud.

Next, to deserialize the telemetry object I wrote the ToMessage method shown in **Figure 9**. In the first step, ToMessage gets the JSON-formatted string, representing the Data object. This is done with JsonConvert.SerializeObject. In the second step, the JSON string is converted to a byte array with Encoding.ASCII.GetBytes. The result of this operation is used to instantiate the Microsoft.Azure.Devices.Client.Message class. Instances of this class can then be sent to the IoT Hub using SendEventAsync method of the DeviceClient class.

Next, I wrote the Generator class to generate and send telemetry data. This class emulates a real sensor reading by pseudo-randomly generating temperature and humidity values (see Data/Generator.cs in the companion source code). To this end, Generator uses the System.Random class to synthesize sensor reading from the given range:

Figure 8 **The DeviceClientHelper Class**

```
public static class DeviceClientHelper
{
  private static readonly string connectionString
    = "<your_connection_string>";

  private static DeviceClient deviceClient;

  public static DeviceClient Init()
  {
    if (deviceClient == null)
    {
      deviceClient = DeviceClient.
        CreateFromConnectionString(connectionString);
    }

    return deviceClient;
  }
}
```

Figure 9 **The Data Class**

```
public class Data
{
  public double Temperature { get; set; }
  public double Humidity { get; set; }

  public Message ToMessage()
  {
    var dataJson = JsonConvert.SerializeObject(this);

    return new Message(Encoding.ASCII.GetBytes(dataJson));
  }

  public override string ToString()
  {
    return $"Temperature: {Temperature,6:F2}, Humidity: {Humidity,6:F2}";
  }
}
```

```
private Random randomNumberGenerator = new Random();

private double GetRandomValue(MeasurementRange measurementRange)
{
  var randomValueRescaled = randomNumberGenerator.NextDouble()
    * measurementRange.ValueRange();

  return measurementRange.Min + randomValueRescaled;
}
```

The measurement range is represented by instances of the MeasurementRange class:

```
public class MeasurementRange
{
  public double Min { get; set; }
  public double Max { get; set; }

  public double ValueRange()
  {
    return Max - Min;
  }
}
```

The Generator class has two fields of the preceding type (MeasurementRange). They correspond to the temperature and humidity measures:

```
private readonly MeasurementRange temperatureRange
  = new MeasurementRange() { Min = -20, Max = 60 };
private readonly MeasurementRange humidityRange
  = new MeasurementRange() { Min = 0, Max = 100 };
```

Note that these ranges are the same as specified previously in the device template.

The Generator class uses an instance of DeviceClient and CancellationToken. The first is used to send the telemetry, while the second will break the infinite telemetry loop. Actual instances of the DeviceClient and CancellationToken are passed through the Generator class constructor:

```
private DeviceClient deviceClient;
private CancellationToken cancellationToken;

public Generator(DeviceClient deviceClient,
  CancellationToken cancellationToken)
{
  Check.IsNull(deviceClient);
  Check.IsNull(cancellationToken);

  this.deviceClient = deviceClient;
  this.cancellationToken = cancellationToken;
}
```

Figure 10 **The TelemetryAction Method**

```
public bool IsTelemetryActive { get; set; } = true;

private void TelemetryAction()
{
  while (!cancellationToken.IsCancellationRequested)
  {
    var telemetryData = new Data()
    {
      Temperature = GetRandomValue(temperatureRange),
      Humidity = GetRandomValue(humidityRange)
    };

    if (IsTelemetryActive)
    {
      deviceClient.SendEventAsync(telemetryData.ToMessage());

      Console.WriteLine($"Sending telemetry: {telemetryData}");
    }
    else
    {
      Console.WriteLine("Idle");
    }
      Task.Delay(delayTime).Wait();
  }
}
```

The Check class is a static helper class used to verify whether arguments are null or not (see in the companion code Helpers/Check.cs).

The telemetry data is generated and sent to the cloud within the while loop implemented within the instance method TelemetryAction of the Generator class, shown in **Figure 10**.

Note that the telemetry is sent to the cloud only when the IsTelemetryActive property is true. This property can be changed on the cloud endpoint using the Settings tab in the IoT Central app (**Figure 4**).

The TelemetryAction is executed in the background using the Task-based asynchronous pattern:

```
public Task Start()
{
  telemetryTask = new Task(TelemetryAction);

  telemetryTask.Start();

  return telemetryTask;
}
```

Putting Things Together

With the DeviceClientHelper and the Generator classes ready, I combined them in the Program class (see Program.cs in the companion code). I started by implementing the static Program.Main method as shown in **Figure 11**.

Program.Main first wires an event handler to the Console.CancelKeyPress event in order to stop the telemetry loop (using the cancellation token) and close the application:

```
private static void CancelKeyPressHandler(object sender,
  ConsoleCancelEventArgs e)
{
  if (e.SpecialKey == ConsoleSpecialKey.ControlC)
  {
    cancellationTokenSource.Cancel();

    Environment.Exit(0);
  }
}
```

Next, the Main method connects to the IoT Hub using static Init method of the DeviceClientHelper. Init returns an instance of the DeviceClient class, which is then passed to the Generator class constructor.

The method SetDesiredPropertyUpdateCallbackAsync of the DeviceClient class instance is now used to set up the

Figure 11 **The Program.Main Method**

```
private static CancellationTokenSource cancellationTokenSource
  = new CancellationTokenSource();

static void Main(string[] args)
{
  // Configure cancel key press handler (to stop the app)
  Console.CancelKeyPress += new ConsoleCancelEventHandler(
    CancelKeyPressHandler);

  // Connect to the cloud
  var deviceClient = DeviceClientHelper.Init();

  // Telemetry generator produces random temperature
  // and humidity, and then sends them both to the cloud
  var telemetryGenerator = new Generator(
    deviceClient, cancellationTokenSource.Token);

  // Associate handler to update device properties according to cloud requests
  deviceClient.SetDesiredPropertyUpdateCallbackAsync(
    PropertyUpdateCallback, telemetryGenerator).Wait();

  // Start telemetry
  telemetryGenerator.Start().Wait();
}
```

Figure 12 **PropertyUpdateCallback**

```
private static readonly string telemetryActivePropertyName =
  "IsTelemetryActive";
private static readonly string propertyValue = "value";

private static Task PropertyUpdateCallback(
  TwinCollection desiredProperties, object userContext)
{
  if (desiredProperties.Contains(telemetryActivePropertyName))
  {
    var telemetryGenerator = userContext as Generator;

    telemetryGenerator.IsTelemetryActive =
      desiredProperties[telemetryActivePropertyName][propertyValue];
  }

  return Task.CompletedTask;
}
```

callback that's invoked when the operator changes device settings at the cloud endpoint. This callback, PropertyUpdateCallback (see **Figure 12**) is provided with an instance of the Microsoft.Azure.Devices.Shared.TwinCollection class. This object represents the collection of device settings. In particular, the class indexer can be used to read values of the selected settings. You identify particular settings using their Field Names, configured at the cloud endpoint. In **Figure 12**, I show how to read the value of the IsTelemetryActive setting and use it to update the corresponding property of the Generator class instance.

Finally, the Main method starts the telemetry by invoking the Start method of the Generator class instance.

To test the client app you need to execute it. If the connection string is valid, the app will connect to the IoT hub and start streaming telemetry data. Each synthesized measurement will be printed in the console and you'll then see these measures in the cloud dashboard (refer back to **Figure 1**). You can also remotely change IsTelemetryActive to temporarily disable telemetry. To do so, open the Settings tab of your Azure IoT Central app, switch the toggle, and click Update button. In that case, the client app prints the Idle string instead of the actual telemetry data.

Wrapping up

In this article I showed how to rapidly create a custom, fully functional, modern-looking Web application for an IoT solution using Azure IoT Central. I also demonstrated how to create a device template and use it to present telemetry data, device location and KPIs. I then developed a C# client app and connected it to stream telemetry data to the cloud. Finally, I showed how to respond to device setting changes requested through the IoT Central application. You can use all this to quickly develop modern Web dashboards for your IoT solutions without any prior cloud or Web programming knowledge. ■

Dawid Borycki *is a software engineer and biomedical researcher, author and conference speaker. He enjoys learning new technologies for software experimenting and prototyping. Borycki is an author of two Microsoft Press books: "Programming for Mixed Reality" (2018) and "Programming for the Internet of Things" (2017).*

Thanks to the following Microsoft technical expert for reviewing this article *Bruno Sonnino*

Using Azure Containers to Provide an On-Demand R Server

Will Stott

In an article in the November 2018 issue of *MSDN Magazine*, "Web Site Background Processing with Azure Service Bus Queues" (msdn.com/magazine/mt830371), I explained how you can use an Azure Function and a Service Bus Queue to handle long-running background processing for your Web site. In this second article, I explain how you can use such processing to start a server when traffic arrives from your Web App and then use the server to perform classification tasks before automatically shutting down the server after the Web traffic has stopped for a prescribed time. I treat the server as a black box because it's implemented as a Docker container.

In my case the server runs Linux and a logistic regression classifier developed in the statistical language R to provide quality control data for ultrasound scans as part of my research for the United Kingdom Collaborative Trial of Ovarian Cancer Screening (UKCTOCS). Your server, however, might run Windows and Python to identify arbitrage opportunities for foreign exchange currency transactions. It really doesn't matter what the server does, so long as it can be made into a Docker image and it has an API you can access.

The key technology you'll encounter in this article is the Azure.Management.Fluent API, which is used to programmatically create and delete an Azure Container Instance (ACI) for your server. In addition, you'll build on your earlier work, as described in the previous article, to create an Azure Function with a timer trigger to orchestrate these actions. You'll also extend the existing Azure Function with the Service Bus Queue trigger so it can pass input data referenced by the queued message to the classifier running in your ACI using its OpenCPU API. In this way you'll employ Azure server-less functions to perform meaningful long-running background tasks for an ASP.NET Core 2.1 MVC Web App. An overview of the entire system is shown in **Figure 1**.

Recreating the project featured in this article requires only rudimentary Web and C# development skills, but assumes that you already built the Azure Functions project described in the previous article. It also assumes that you built the simple WebApp and database project described in the companion online resource. You can find the resource PDF, related source code and instructions for building the solution shown in **Figure 1** at github.com/wpqs/MSDNOvaryVis. In addition, the Docker Image containing the server's R classifier is freely available from hub.docker.com as r/wpqs/ovary-classifier. In terms of tools, you'll need Visual Studio 2017 v15.7 with .NET Core 2.1 SDK and the Web development workload, as well as v15.0.404240 of the Azure Function and WebJob tools. Note that the Community version of Visual Studio is available for free. You'll also need an Azure Subscription, but again, you can get what you need for free if you're a new customer.

This article discusses:

- Creating and deleting Azure Container Instances from a Docker image using C# and Azure.Management.Fluent
- Using an Azure Function to implement periodic processing
- Provisioning resources using the Azure Portal PowerShell Console

Technologies discussed:

Azure Container Instances, Azure Functions, Docker, C#, Visual Studio 2017

Code download available at:

github.com/wpqs/MSDNOvaryVis

Running Your Server in an Azure Container Instance

Anyone who has provisioned a virtual machine (VM) will be familiar with the idea of loading its image from some sort of file. This image contains the OS, settings, data, application software and everything else needed by your VM. Docker allows you to do much the same thing, but more efficiently because resources are shared better between instances of images running on the same machine.

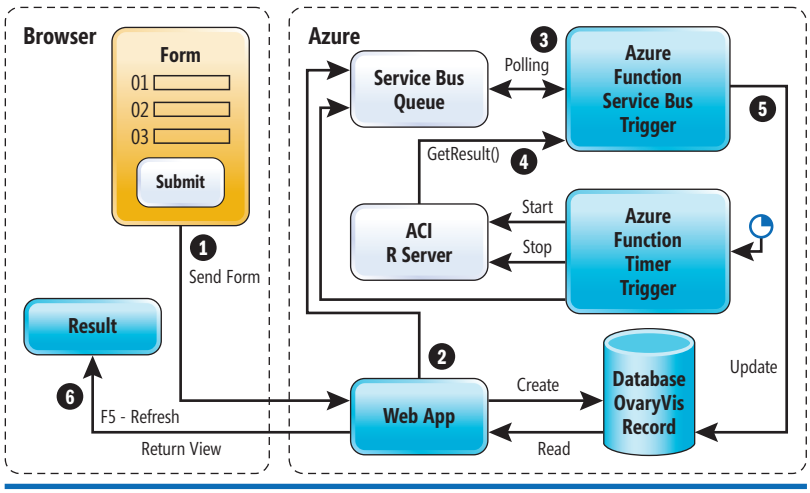


Figure 1 **System Overview**

The Docker image used for my research was based on a Linux server with the R statistical software and OpenCPU installed. OpenCPU (opencpu.org) provided me with an API so my R classifier function could be invoked using HTTP messaging through the host machine's port 80. The people at OpenCPU also helpfully provided a public Docker image that served as the base for my server. All I needed to do was add the R function and my classifier.

This ability to build up Docker images layer upon layer and reuse existing work is an important part of what makes it so attractive to developers. Azure provides support for Docker by allowing you to create an Azure Container Instance (ACI), which is like a Docker Container in that it's an instance of a Docker image. Therefore, by creating an ACI from the image in Docker Hub, you start an instance of the server on the Azure platform and when you delete the ACI this server stops.

Preparatory Work

Before implementing the code needed to manage an ACI, it's necessary to do some preparatory work by installing a few additional packages in the Azure Function App project developed as part of the work described in the previous article. You'll also need to create a Security Principal for this Azure Function and update its Application Settings.

Checking the existing Azure Function App project and its packages is a good idea once you've opened the Visual Studio MSDNOvaryVis Solution, either from the article's download or as created when following the instructions in my previous article.

The key packages to check using the NuGet Package Manager are Microsoft.EntityFrameworkCore.SqlServer v2.1.2 and Microsoft.NET.Sdk.Functions v1.0.19. These packages were used for this article, but you might want to try later versions for your own work. You can find details about the use of NuGet Package Manager in the related article I've hosted online in the GitHub repository.

Adding the packages needed to control Azure Containers and send messages to your ServiceBus Queue just requires you to give the following commands from the package manager:

```
Install-Package Microsoft.Azure.Management.Fluent
  -Project OvaryVisFnApp
  -Version 1.14.0
Install-Package Microsoft.Azure.WebJobs.ServiceBus
  -Project OvaryVisFnApp
  -Version 3.0.0-beta8
Install-Package Microsoft.Azure.ServiceBus
  -Project OvaryVisFnApp
  -Version 3.1.0
```

Adding these packages means that you need to set the project's target framework to netstandard2.0.

Granting your Azure Function the authority it needs to create and delete an Azure Container Instance requires you to create a security principal for your Azure Subscription. The PowerShell Console built into the Azure Portal Web site, shown in **Figure 2**, provides an easy way to do this using the following commands, though you need to replace EEE with a suitable password and MsdnOvaryVis with the name of your own Azure subscription:

> You can add settings by issuing commands from the Cloud Shell.

```
az account set --subscription MsdnOvaryVis
$password = ConvertTo-SecureString "EEE" -AsPlainText -Force
$sp = New-AzureRmADServicePrincipal -DisplayName "MSDNOvaryVisApp"
  -Password $password
New-AzureRmRoleAssignment -ServicePrincipalName $sp.ApplicationId
  -RoleDefinitionName Contributor
$sp | Select DisplayName, ApplicationId
Get-AzureRmSubscription -SubscriptionName MsdnOvaryVis
```

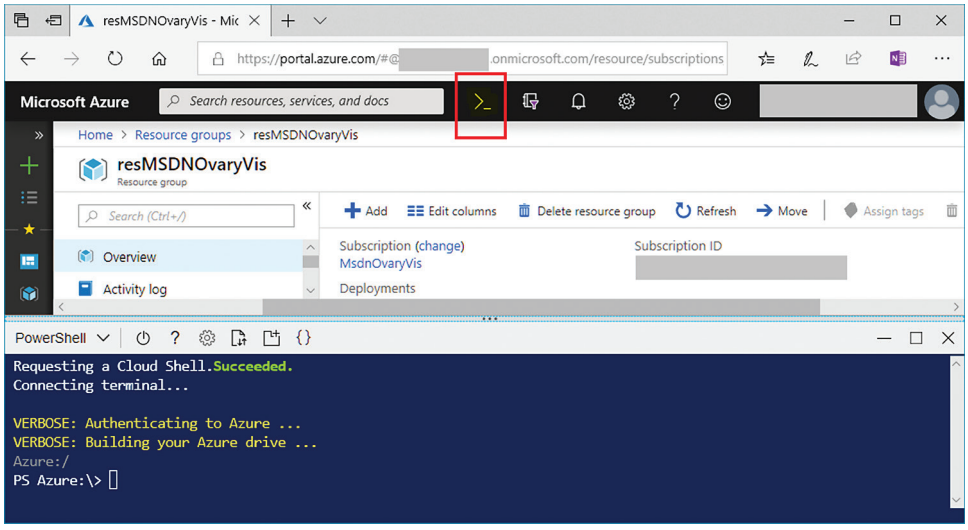


Figure 2 **The Cloud Shell of the Azure Portal PowerShell Console**

You should copy the responses and your password into Notepad (or similar) so you can save the information. The important values to keep are your password (EEE), App Id and Tenant Id. However, instead of hardcoding them into your code, it's better to reference them from the Application settings of your Azure Function App service. In a production system you might consider using the new Azure Managed Service Identity (MSI) as an alternative to creating a security principal and adding it to the application settings of your Azure Function as described earlier.

Updating the Application settings of your Azure Function App allows you to avoid hardcoding changeable or sensitive values into your code. You can open its Application Settings blade in the Azure Portal and then add the items in the red boxes shown in **Figure 3**. Alternatively, you can add settings by issuing commands from the Cloud Shell. For example, given my Function App is called MSDNOvaryVisFnApp and is located in the resMSDNOvaryVis resource group, I can give its Security Principal ID setting a value of DDD, as follows:

```
az functionapp config appsettings set --name MSDNOvaryVisFnApp
  --resource-group resMSDNOvaryVis --settings 'SecPrincipalId=DDD'
```

A complete list of the commands needed to apply the required application settings can be found in the AzureResCmds.txt file in the download supplied with this article.

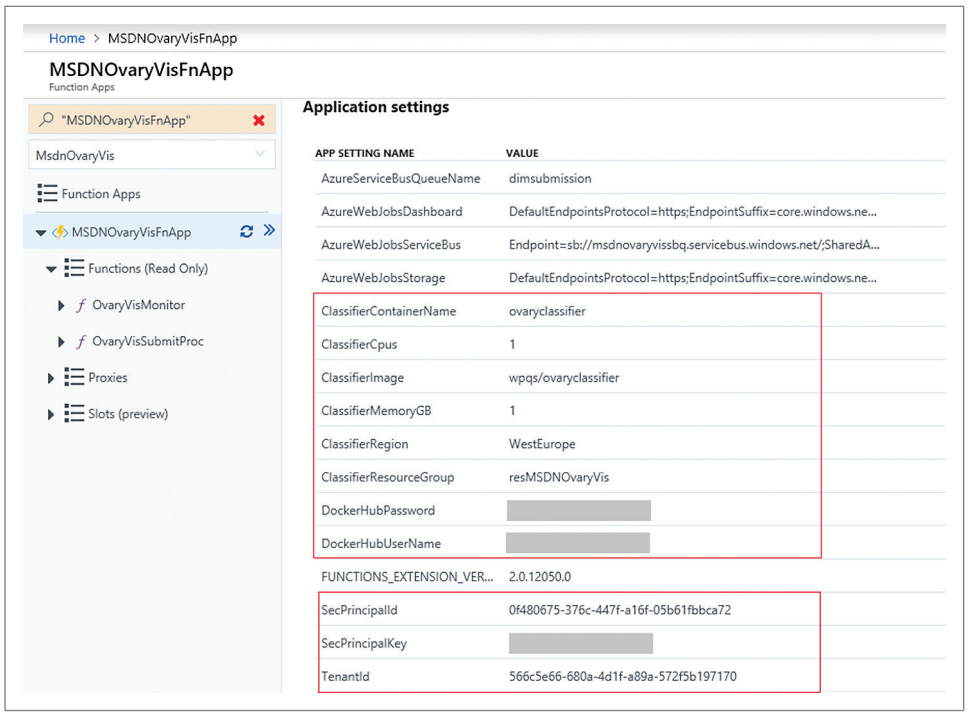


Figure 3 **Azure Function App Service Application Settings**

GetContainerGroup allows you to find any container group already created in Azure.

Creating the Server Class

It makes sense to add a class to your Azure Function App project so you can put all the methods needed to manage your server in one place, specifically:

- GetAzure returns an interface initialized with the security principal you created earlier.
- GetContainerGroup finds an existing ACI running in your Azure resources.
- IsRunning returns a Boolean to signal whether your server is currently running.
- StartAsync creates an ACI using the specified Docker image
- GetResult Async passes the input data to your server and return the result.
- StopAsync deletes an ACI.

You can use Visual Studio to add an empty C# class suitable for this work by selecting your OvaryVisFnApp project, right-clicking

Figure 4 **Implementation of Server.StartAsync Starts a Server On-Demand**

```
public static async Task<string> StartAsync(IConfigurationRoot config)
{
  string rc = "server start";
  if (await Server.IsRunning(config) == true )
    rc += " completed, already running";
  else
  {
    var region = config["ClassifierRegion"];
    var resourceGroupName = config["ClassifierResourceGroup"];
    var containerName = config["ClassifierContainerName"];
    var classifierImage = config["ClassifierImage"];
    var cpus = config["ClassifierCpus"];
    var memory = config["ClassifierMemoryGB"];
    var dockeruser = config["DockerHubUserName"];
    var dockerpwd = config["DockerHubPassword"];

    double.TryParse(cpus, out double cpuCount);
    double.TryParse(memory, out double memoryGb);

    var containerGroup =
      await GetAzure(config).ContainerGroups.Define(containerName)
        .WithRegion(Region.Create(region)).WithExistingResourceGroup(
          resourceGroupName)
        .WithLinux().WithPrivateImageRegistry("index.docker.io",
          dockeruser, dockerpwd)
        .WithoutVolume().DefineContainerInstance(containerName)
        .WithImage(classifierImage).WithExternalTcpPort(80)
        .WithCpuCoreCount(cpuCount).WithMemorySizeInGB(memoryGb)
        .Attach().WithRestartPolicy(
          ContainerGroupRestartPolicy.OnFailure).CreateAsync();
    _classifierIP = containerGroup?.IPAddress ?? "";
    if ((_classifierIP == null) || (_classifierIP.Length == 0))
      rc += " failed";
    else
      rc += " completed ok";
  }
  return rc;
}
```

and choosing Add | Class. This opens the Add New Item dialog box with C# Class selected. Just type Server.cs as its file name to create the corresponding Server class. Let's walk through implementing each of these methods.

GetAzure supports the initialization of the Security Principal you created earlier, so that your code has the permissions it needs to manage resources for your Azure subscription. It's implemented as follows:

```
private static IAzure GetAzure(IConfigurationRoot config)
{
  AzureCredentials credentials = SdkContext.AzureCredentialsFactory
    .FromServicePrincipal(config["SecPrincipalId"],
    config["SecPrincipalKey"], config["TenantId"],
    AzureEnvironment.AzureGlobalCloud);
  IAzure rc = Azure.Configure().WithLogLevel(
    HttpLoggingDelegatingHandler.Level.Basic)
    .Authenticate(credentials).WithDefaultSubscription();
  return rc;
}
```

GetContainerGroup allows you to find any container group already created in Azure, so you don't attempt to create a new ACI when one already exists. It's simply a matter of listing the container groups currently present in your Resource Group and returning the one with the container name specified in your application settings, or null if none are found. It's implemented as follows:

```
private static async
Task<IContainerGroup>GetContainerGroup(IConfigurationRoot config)
{
  IContainerGroup rc = null;
  var classifierResoureGroup = config["ClassifierResourceGroup"];
  var list = await azure.ContainerGroups.ListByResourceGroupAsync(
    classifierResoureGroup);
  foreach (var container in list){
    if((rc = await GetAzure(config).ContainerGroups
        .GetByResourceGroupAsync(
          classifierResoureGroup,config["ClassifierContainerName"]))
      break;
  }
  return rc;
}
```

IsRunning provides a convenient way to determine whether your server is currently running. It also sets the variable _classifierIP to the IP address of the server if running. You implement it like so:

```
private static string _classifierIP = "";
public static string GetIP() { return _classifierIP; }
public static async Task<bool> IsRunning(IConfigurationRoot config)
{
  _classifierIP = (await
GetContainerGroup(config)).IPAddress
?? "";
  return (_classifierIP.Length > 0)
? true : false;
}
```

StartAsync lets you create an Azure Container Instance using the r/wpqs/ovaryclassifier Docker image to provision and start a server on-demand. It's implemented in the Server class as shown in **Figure 4**. You'll see that the server is started with TcpPort 80 open so you can communicate with it using the HTTP messaging used by OpenCPU API. You'll also see that various parameters are passed to the method from the Azure Function's application settings as shown in the top red box in **Figure 3**. These are the name and password for your Docker account, the container image, and the required specification of the ACI in terms of the minimum number of processor cores and memory size. Further information about the options for creating an Azure Container Instance can be found in Microsoft Docs at bit.ly/2CDKrJg.

GetResultAsync allows you to pass data between your Azure Function and the Server. The exact implementation will depend on the way your server has implemented its API. In this case the OpenCPU API is used to pass the ovary dimensions to the classifier and get back the result—ovary visualized or not. This particular API was built to allow HTTP messaging with servers hosting R statistical functions, so it needs the following type of method:

```
public static async Task<int> GetResultAsync(int D1mm, int D2mm, int D3mm)
{
  int rc = -99;
  var ip = Server.GetIP();
  if (ip != "")
  {
  // Post message to server with content formed
  // from KeyValue pairs from input params
  // Check response and then query the server for the result
  // Process the result to get the return value:
  // 0 is not visualized, 1 is visualized
  }
  return rc;
}
```

The full implementation of GetResultAsync is available in the source-code download for this article, but it's unlikely to be useful to you unless your server also implements the OpenCPU API.

StopAsync enables you to delete an ACI, thereby stopping the associated Azure server. This is implemented by adding a further method to the Server class, like so:

```
public static async Task<string> StopAsync(IConfigurationRoot config)
{
  _classifierIP = "";
  IContainerGroup containerGroup = await GetContainerGroup(config);
  if (containerGroup == null)
    return  "server stop completed already";
  else
  {
    await GetAzure(config).ContainerGroups.DeleteByIdAsync(containerGroup.Id);
    return " server stop completed ok";
  }
}
```

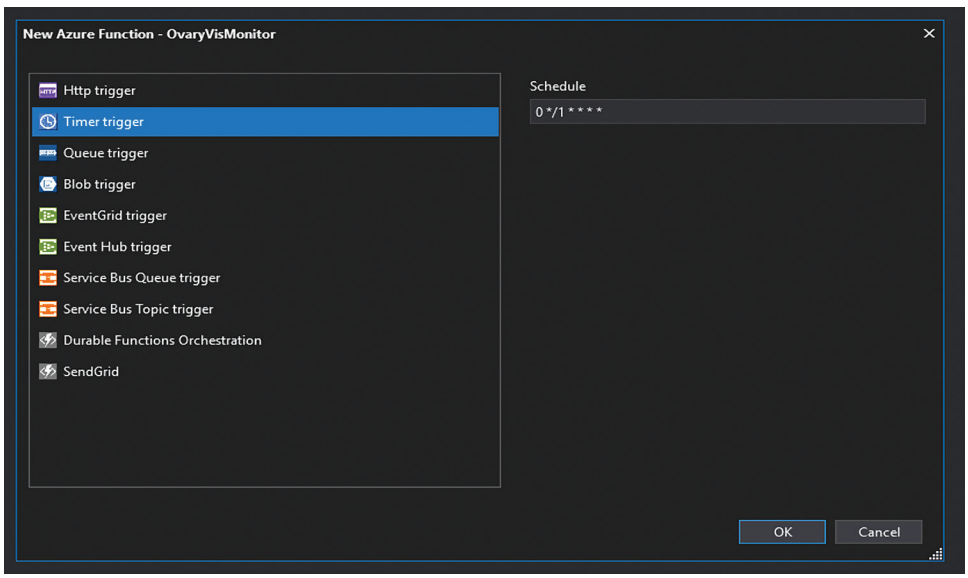


Figure 5 **New Azure Function Dialog Box**

Implementing the Azure Function with Timer Trigger

An Azure Function with a periodic timer trigger is a good way to manage the starting of the server, as this operation may take a number of minutes to complete, potentially preventing timely processing of messages if performed in the Service Bus Queue Trigger Azure Function. Last month's article described how to create the Visual Studio project you need to contain such a function. If you're following along with the code, you just need to select the OvaryVisFnApp project, right-click to select New Azure Function and then specify the name of your new file as OvaryVisMonitor.cs. This opens the dialog box shown in **Figure 5**, which lets you select a Timer Trigger and set its frequency using a CRON expression—in this case, every minute. When you click OK, the dialog box closes and a new Azure Function is created in a class called OvaryVisMonitor, with a Run method appropriate for your selections.

Once you've created this class you should add the following code just above the Run method in order to initialize the client for your Service Bus Queue:

```
private static IQueueClient _queueClient = null;
private static readonly object _accesslock = new object();
private static void SetupServiceBus(string connection, string queueName)
{
  lock (_accesslock) {
    if (_queueClient == null)
      _queueClient = new QueueClient(connection, queueName);
  }
}
```

You'll see that no matter how many times this SetupServiceBus method is called it only creates the Service Bus Queue client once and has a lock to make it thread safe. Therefore, you can safely do this as shown in **Figure 6**, knowing that even though SetupServiceBus is invoked each time your periodic timer function runs, the client will only be instantiated on the first call.

You should note that your Run method needs the addition of the exeContext parameter, as shown in **Figure 6**. It's used to initialize the config variable, which allows access to the Azure Function Application Settings, as provided in the earlier preparatory work. You should also note the use of the Mutex expression, which stops the body of the function being re-entered should the Run function be invoked again before its previous invocation is complete. This prevents any attempt to create a second Azure Container Instance should the startup of your server take more than a minute.

The operation of the Run function depends on obtaining a list of OvaryVis database records that have not yet been processed (that is, records with their ResultVis field set as -1). If this list isn't empty and the server isn't running, then Server.StartAsync is called, which blocks any further progression of the Run method code for a significant period; perhaps several minutes. Once Server.StartAsync returns, the Run method completes and the Mutex is released. Therefore, upon its next invocation, one or more messages will be sent to your Service Bus Queue—one for each record in the pendingJobs list. This allows your pending jobs to be processed if they had been sent by OvaryVisWebApp, as described in the first article.

The final part of the timer function is concerned with stopping the server after 10 minutes of inactivity. You can see that when all the records in the OvaryVis table are at least 10 minutes old, whether processed or not, the code calls Server.StopAsync.

Updating the FormSubmittedProc method used by the OvaryVisSubmitProc Azure function completes your code implementation. You just need to edit the code created for the previous article so that the FormSubmittedProc method (called by Run) is changed, as shown in **Figure 7**.

Reviewing the code in **Figure 7**, you'll see that FormSubmittedProc checks whether the server is running, and if so, calls GetResultAsync to obtain the result from the classifier. The database record is then updated so the result of the classifier is set in the ResultVis field. A value of 0 means ovary not visualized and 1 means ovary visualized. The StatusMsg field is also updated so the progress of the operation can be shown whenever the user refreshes the browser (F5). If the server isn't running, the record's StatusMsg field is updated with an appropriate message so the user knows to wait for the event to be resubmitted by OvaryVisMonitor as described earlier.

Building and publishing your updated OvaryVisFnApp project is done as described in the previous article: Right-click, select Publish and click the Publish button. However, make sure that beforehand you use

Figure 6 **Implementation of the OvaryVisMonitor Timer Trigger Azure Function**

```
[FunctionName("OvaryVisMonitor")]
public static async Task Run([TimerTrigger("0 */1 * * * *")]TimerInfo myTimer,
  TraceWriter log, Microsoft.Azure.WebJobs.ExecutionContext exeContext)
{
  using (Mutex mutex = new Mutex(true, "MSDNOvaryVisMonitorMutuex", out bool doRun))
  {
    if (doRun == true)
    {
      var config = new ConfigurationBuilder().SetBasePath(
        exeContext?.FunctionAppDirectory)
        .AddJsonFile("local.settings.json", optional: true, reloadOnChange: true)
        .AddEnvironmentVariables().Build();
      var optionsBuilder = new DbContextOptionsBuilder<ApplicationDbContext>();
      optionsBuilder.UseSqlServer(
        config["ConnectionStrings:DefaultConnection"]);
      ApplicationDbContext dbContext =
        new ApplicationDbContext(optionsBuilder.Options);
      DateTime expiry = DateTime.UtcNow.AddMinutes(-10);
      var pendingJobs = await dbContext.OvaryVis
        .Where(a => (a.ResultVis == -1) && (a.JobSubmitted > expiry)).ToListAsync();
      if (pendingJobs.Count > 0)
      {
        if (await Server.IsRunning(config) == false)
          await Server.StartAsync(config);
        else
        {
          SetupServiceBus(config["AzureWebJobsServiceBus"],
                          config["AzureServiceBusQueueName"]);
          foreach (var job in pendingJobs)
          {
            var message = new Message(Encoding.UTF8.GetBytes(job.Id));
            await _queueClient.SendAsync(message);
          }
        }
      }
      else
      {
        if (await Server.IsRunning(config) == true)
        {
          var recentJobs = await dbContext.OvaryVis
            .Where(a => a.JobSubmitted > expiry).ToListAsync();
          if (recentJobs.Count == 0)
            await Server.StopAsync(config);
        }
      }
    }
  }
}
```

Figure 7 **Updated FormSubmittedProc Method in Existing Azure Function**

```
private static async Task<string> FormSubmittedProc(IConfigurationRoot config,
  ApplicationDbContext dbContext, string queueItem)
{
  string rc = "FormSubmittedProc: ";
  if (queueItem != null)
  {
    var rec = await dbContext.OvaryVis.SingleOrDefaultAsync(a => a.Id == queueItem);
    if (rec == null)
      rc += string.Format("record not found: Id={0}", queueItem);
    else
    {
      if ((rec.ResultVis != -1) || (rec.ResultVis == -99))
        rc += string.Format("already processed: result={0}", rec.ResultVis);
      else
      {
        if (await Server.IsRunning(config) == false)
          rc += "server not running, wait for job to be resubmitted";
        else
        {
          rec.ResultVis = await Server.GetResultAsync(
            rec.D1mm, rec.D2mm, rec.D3mm);
          if (rec.ResultVis < 0)
            rc += string.Format("server running result={0} - error",
              rec.ResultVis);
          else
          {
            rc += string.Format("server running result={0} - success (ovary {1})",
                  rec.ResultVis, (rec.ResultVis == 1) ? "found" : "not found");
          }
        }
        rec.StatusMsg = rc;
        dbContext.Update(rec);
        await dbContext.SaveChangesAsync();
      }
    }
  }
  return rc;
}
```

the Azure Portal to stop your Azure Function App service by issuing the following command from the Cloud Shell, as shown in **Figure 2**:

```
az functionapp stop --name MSDNOvaryVisFnApp
  --resource-group resMSDNOvaryVis
```

Once published you need to restart the service and then functionally test it, paying particular attention to watching the Functions Server App log, as well as checking your Azure Resource Group blade where the Container Instance will appear when the server is running.

Basic functional testing can be performed by opening your browser at the WebApp's URL as displayed in its Overview blade of the Azure Portal. Once the homepage appears, enter a set of ovary dimensions and then click Submit. You'll immediately be redirected to the Results page, where you'll see the values of the database record relating to your submission, as shown in the left side of **Figure 8**. Repeatedly pressing F5 will cause the page to be updated with the current values of this record. Eventually the results of the classifier's assessment for these ovary dimensions will be shown, as displayed in the right side of **Figure 8**. If the server is already running, this will take only a few seconds, otherwise it may take a few minutes. After 10 minutes of inactivity on your Web site, the server will automatically close down, demonstrating the on-demand nature of its implementation.

Wrapping Up

The features of the Web site developed here are trivial. It just collects three dimensions from a form in order to produce a binary response: ovary visualized or not. However, its implementation is far from trivial. In the previous article, I showed you how to implement a way of processing the data in the background using an Azure Service Bus Queue and an Azure Function App service. In this article, I've extended this background processing mechanism to create an ACI from a Docker Image published on Docker Hub. This image contains a Linux server hosting R and a custom logistic regression classifier, as well as the OpenCPU API that allows communication with the server using HTTP messaging. What's more, I implemented the logic to create this ACI on-demand, starting the server only when needed and shutting it down after 10 minutes of inactivity.

You could easily improve the system presented here by adding an Azure SignalR Service to automatically update your result Web page instead of relying on the user to periodically press F5 to refresh it. You might also consider extending the way messages are handled so that multiple ACIs are created when the Web site is experiencing high load situations. However, this extra complexity wasn't warranted for an article that shows how to implement a server that costs just $1.00 per month (compared to the $50 a month you might pay for a virtual server operating 24/7). Even with the cost of the database service, you're looking at paying less than $5.00 a month. That's good value considering the technology you're employing, but more importantly, it's a robust solution to a real-world problem that you can build without too much effort. ■

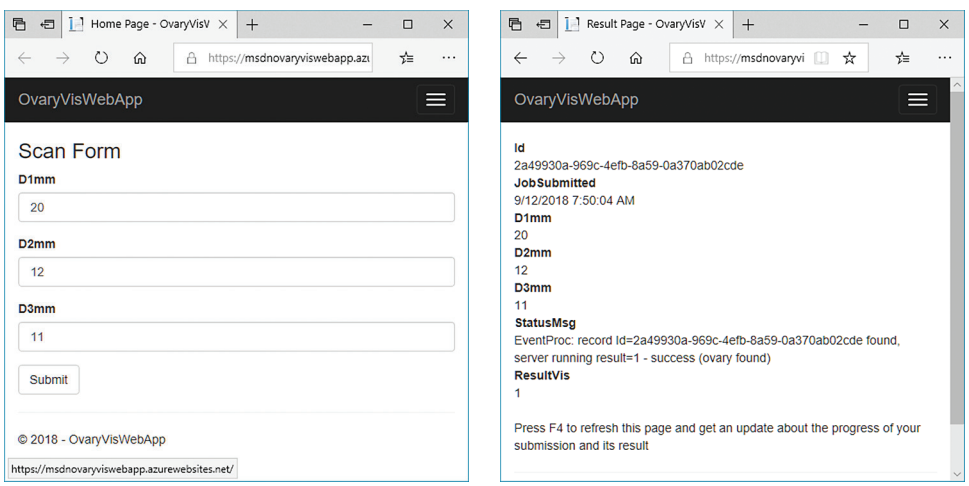


Figure 8 **Functional Test**

Dr. Will Stott *has more than 25 years of experience working as a contractor/consultant for a wide range of companies in the United Kingdom and Europe. For the last 10 years most of his time has been spent doing research at UCL on ovarian cancer screening. Dr. Stott has spoken at many conferences and is the author of papers published in various journals, as well as the book "Visual Studio Team System: Better Software Development for Agile Teams" (Addison-Wesley Professional, 2007).*

Thanks to the following Microsoft technical expert for reviewing this article: *Srikantan Sankaran*

DINO ESPOSITO

Blazor Custom Components

The Blazor programming experience leads naturally toward an extensive use of components. In Blazor, a component is a .NET class that implements some UI rendering logic based on some state. Blazor components are close to the idea of Web components from the upcoming W3C specification and analogous implementations in single-page application (SPA) frameworks. A Blazor component is a combination of HTML, C# and interoperable JavaScript code and CSS that together act as a single element with a common programming interface. A Blazor component can fire events and expose properties in much the same way an HTML DOM element does.

In my recent article on Blazor, I presented a couple of dedicated components. The first provided a comprehensive UI to type some text, run a query and fire an external event with retrieved data as an argument. The second component in the example acted as a made-to-measure, non-customizable grid that would handle the event, grab the data and populate the internal table. The input field collecting the query text offered auto-completion via the type-ahead Bootstrap extension. In this article, I build from that point and discuss the design and implementation of a completely Blazor-based component for type-ahead functionality. At the end of the task, you'll have a reusable CSHTML file with no dependencies on anything but the core Bootstrap 4 bundle.

The Public Interface of TypeAhead

The goal is to produce a Blazor-native component that behaves like an extended version of the classic typeahead.js component (see twitter.github.io/typeahead.js). The final component lies in the middle of a textbox and a dropdown list. As users type in the text input field, the component queries a remote URL and gets hints of possible values to enter. The JavaScript typeahead provides suggestions but doesn't force users to accept any of the suggestions. In other words, custom text different from suggestions is still acceptable.

In some cases, though, you want users to enter text and then select from a list of valid entries. Think, for example, of an input field where you need to provide a country name or a customer name. There are more than 200 hundreds countries in the world and often hundreds (or more) customers in a software system. Should you really use a dropdown list? A smarter type-ahead component could let users type in, say, the word "United," and then present a list of matching countries like United Arab Emirates, United Kingdom and United States. If any free text is typed, the code automatically clears the buffer.

Code download available at bit.ly/2PtrhN1.

Another related issue: When free text isn't an option, you likely need to also have a code. You ideally want to type the country or customer name and have posted from the hosting form the ID of the country or the unique identifier for the customer. In pure HTML and JavaScript, you need some extra script that adds a hidden field and manages selections from the typeahead plug-in. A Blazor native component will have all these capabilities hidden under the hood. Let's see how to write such a component.

Design of the Component

The Typeahead component is meant to be an additional input field to use within an HTML form. It's made of a couple standard input fields, one of type text and one hidden. The hidden input field will feature a NAME attribute that will make it fully interoperable if used within a form. Here's some sample markup for using the new component:

```
<typeahead style="margin-top: 40px;"
           class="form-control"
           url="/hint/countries1"
           selectionOnly="true"
           name="country"
           placeholder="Type something"
           onSelectionMade="@ShowSelection" />
```

Figure 1 **Markup of the Typeahead Component**

```
<div class="blazor-typeahead-container">
  <div class="input-group">
    <input type="text" class="@Class" style="@Style"
           placeholder="@Placeholder"
           oninput="this.blur(); this.focus();"
           bind="@SelectedText"
           onblur="@(ev => TryAutoComplete(ev))" />
    <input type="hidden" name="@Name" bind="@SelectedValue" />

    <div class="input-group-append">
      <button class="btn btn-outline-secondary dropdown-toggle"
              type="button" data-toggle="dropdown"
              style="display: none;">
      </button>
      <div class="dropdown-menu dropdown-menu-right
                  scrollable-menu @(_isOpen ? "show" : "")"
           style="width: 100%;">
        <h6 class="dropdown-header">@Items.Count item(s)</h6>
        @foreach (var item in Items)
        {
          <a class="dropdown-item"
           onclick="@(() => TrySelect(item))">
            @((MarkupString) item.MenuText)
          </a>
        }
      </div>
    </div>
  </div>
</div>
```

Figure 2 **Properties of TypeAhead Component**

Name	Description
Class	Gets and sets the collection of CSS classes to be applied to the internal HTML elements of the component.
Name	Gets and sets the value of the NAME attribute when the component is incorporated in an HTML form.
Placeholder	Gets and sets the text to serve as a placeholder for the HTML elements being rendered.
SelectedText	Gets and sets the display text. This serves as both the initial value and the selected text, whether typed by the user or picked from a dropdown menu.
SelectedValue	Gets and sets the selected value. This may or may not match the value of SelectedText. The selected value is bound to the hidden field, whereas SelectedText is bound to the text field.
SelectionOnly	Boolean value that determines whether free text is allowed or users are forced to only select one of the provided hints.
Style	Gets and sets the collection of CSS styles to be applied to the internal HTML elements of the component.
Url	Address of the remote endpoint to call for hints.

As you can see, the component mirrors some HTML-specific attributes such as Style, Class, Name and Placeholder side-by-side with custom attributes such as Url, SelectionOnly and events like onSelectionMode. The Url attribute sets the remote endpoint to call for hints, while the Boolean SelectionOnly attribute controls the behavior of the component and whether input should only come from a selection or if freely typed text is allowed as input. Let's take a look at the Razor markup and related C# of the component in **Figure 1**. You'll find full details in the typeahead.cshtml file of the sample project at bit.ly/2v0lYYZ.

> It's important to note that the HTML layout of the Typeahead component is more complex than just a couple of input fields.

Figure 2 lists the properties defined and supported by the component.

It's important to note that the HTML layout of the Typeahead component is more complex than just a couple of input fields. It's designed to receive hints in the form of a TypeAheadItem class defined as shown here:

```
public class TypeAheadItem
{
  public string MenuText { get; set; }
  public string Value { get; set; }
  public string DisplayText { get; set; }
}
```

Any suggested hint is made of a display text (such as the country name) that sets the input field, a menu text (such as a richer HTML-based text) that appears in the dropdown list, and a value (such as the country code) that uniquely identifies the selected item. The value attribute is optional, but serves a crucial purpose in case the Typeahead component is used as a smart dropdown list. In this case, the role of the Value attribute is the same as the value attribute of the HTML Option element. The code located at the remote endpoint referenced by the Url attribute is expected to return an array of TypeAheadItem entities. **Figure 3** provides an example of an endpoint that returns the list of country names that match a query string.

Figure 3 **Returning a List of Country Names**

```
public JsonResult Countries(
  [Bind(Prefix = "id")] string filter = "")
{
  var list = (from country in CountryRepository().All();
    let match =
      $"{country.CountryName} {country.ContinentName}".ToLower()
    where match.Contains(filter.ToLower())
    select new TypeAheadItem()
    {
      Value = country.CountryCode,
      DisplayText = country.CountryName,
      MenuText = $"{country.CountryName} <b>{country.ContinentName}</b>
        <span class='pull-right'>{country.Capital}</span>"
    }).ToList();

  return Json(list);
}
```

There are a couple of things to notice here and both have to do with expressivity. There should be some sort of intimacy between the hint server and the type-ahead component. As a developer, you have the final word about that. In the sample code discussed in this article, the Web service endpoint returns a collection of TypeAheadItem objects. In a customized implementation, though, you can have the endpoint return a collection of application-specific properties and have the component decide dynamically which property should be used for the text and which for the value. The TypeAheadItem object, however, supplies a third property—MenuText—that contains an HTML string to be assigned to the dropdown list item, as shown in **Figure 4**.

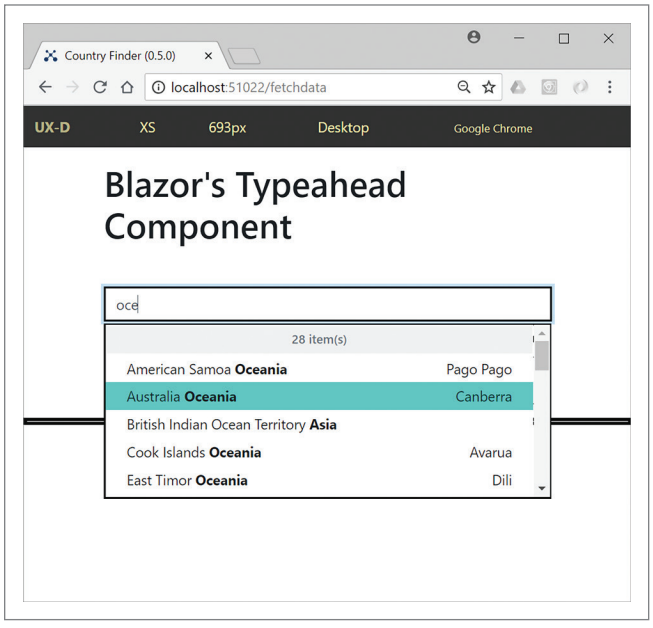


Figure 4 **Typeahead Blazor Component in Action**

The MenuText is set to the concatenation of country name and continent name, along with the name of the capital right justified in the control. You can use any HTML formatting that suits your visual needs.

Having the markup statically determined on the server isn't ideal, however. A much better approach would be sending any relevant data to the client and let the markup be specified as a template parameter to the component. Not coincidentally, templated components are a cool upcoming feature of Blazor that I'll cover in a future column.

In addition, note that in **Figure 4** the user is typing the name of a continent and receives hints for all the countries in that continent. The implementation of the Countries endpoint, in fact, matches the query string ("oce" in the screenshot) against the concatenation of country and continent name, as in the LINQ variable match you see in the code snippet above.

This is the first step of an obviously more powerful way to search for related data. You can, for example, split the query string by commas or spaces to obtain an array of filter strings and then combine them by OR or AND operators. Yet another improvement is the template of the drop-down list item that in the current example is hard-coded in the endpoint implementation, but can be provided as an HTML template along with the query string.

The bottom line is that the Typeahead component presented here uses the Twitter JavaScript typeahead plugin only as a starting point. The Blazor components allow developers to easily hide implementation details, ultimately raising the abstraction level of the code that Web developers write.

The Mechanics of the Component

In **Figure 1**, you have examined the markup behind the Blazor Typeahead component. It relies on Bootstrap 4 and counts on a few custom CSS styles defined in the same CSHTML source file. If you like developers to customize those styles, you simply provide documentation for them. At any rate, developers willing to use the Typeahead component don't have to know about custom CSS styles such as scrollable-menu.

The component is articulated as a Bootstrap 4 input group made of a text input field, a hidden field, and a drop-down button. The text input field is where the user types any query string. The hidden field is where the value of the accepted hint is stored to be forwarded through any host HTML form. Only the hidden field has the HTML name attribute set. The drop-down button provides the menu with hints. The list of menu items is populated any time new text is typed into the input field. The button is not visible by default but its drop-down window is programmatically shown whenever needed. This is done by leveraging Blazor's data binding capabilities. An internal Boolean variable is defined (_isOpen) that determines whether the "show" CSS class of Bootstrap 4 should be added to the drop-down section of the button. Here's the code:

```
<div class="dropdown-menu
            dropdown-menu-right
            scrollable-menu
            @(_isOpen ? "show" : "")">
 ...
</div>
```

The Blazor bind operator is used to bind the SelectedText property to the value property of the text input field and the SelectedValue property to the value property of the hidden field.

How do you trigger the remote query for hints? In plain HTML 5, you would define a handler for the input event. The change event is not appropriate for text boxes as it triggers only once the focus has been lost, which doesn't apply in this particular case. In the version of Blazor used for the article (version 0.5.0), you can attach some C# code to the input event, but it doesn't really work as expected yet.

As a temporary solution, I attached the code that populates the drop-down to the blur event and added some JavaScript code to handle the input event that just calls blur and focus at the DOM level. As you can see in **Figure 1**, the TryAutoComplete method that runs in response to the blur event places the remote call, grabs a JSON array of TypeAheadItem objects, and populates the internal Items collection:

```
async Task TryAutoComplete(UIFocusEventArgs ev)
{
  if (string.IsNullOrWhiteSpace(SelectedText))
  {
    Items.Clear();
      _isOpen = false;
    return;
  }

  var actualUrl = string.Concat(Url.TrimEnd('/'), "/", SelectedText);
  Items = await HttpExecutor.GetJsonAsync<IList<TypeAheadItem>>(actualUrl);
    _isOpen = Items.Count > 0;
}
```

When this happens the dropdown is populated with menu items and displayed, like so:

```
@foreach (var item in Items)
{
  <a class="dropdown-item"
    onclick="@(() => TrySelect(item))">
    @((MarkupString) item.MenuText)
  </a>
}
```

Note the cast to MarkupString that is the Blazor counterpart to Html.Raw in ASP.NET MVC Razor. By default any text processed

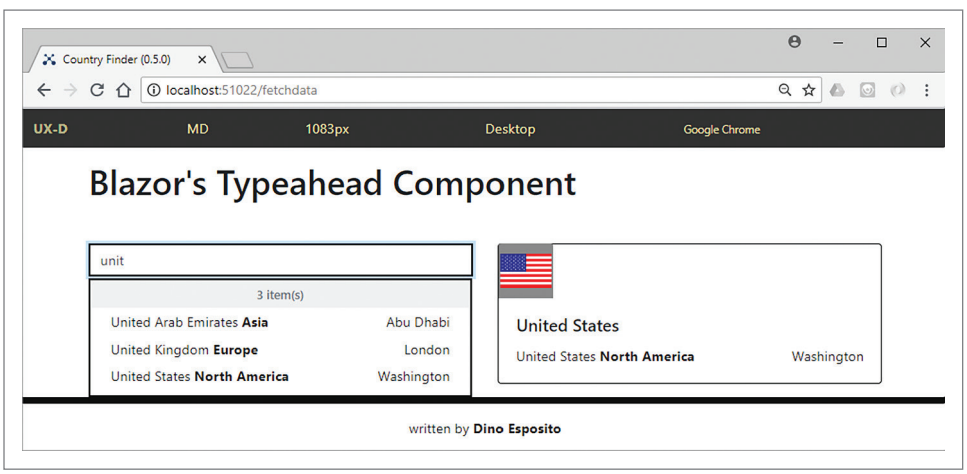


Figure 5 **The Updated View**

by Razor is encoded, except when the expression is cast to the MarkupString type. Hence, if you want HTML to be displayed, you must pass through the MarkupString cast. Every time a menu item is clicked, the method TrySelect runs, like so:

```
void TrySelect(TypeAheadItem item)
{
  _isOpen = false;
  SelectedText = item.DisplayText;
  SelectedValue = item.Value;
  OnSelectionMade?.Invoke(item);
}
```

The method receives the TypeAheadItem object associated with the clicked element. Next, it closes the drop-down by setting _isOpen to false and updates SelectedText and SelectedValue as appropriate. Finally, it calls StateHasChanged to refresh the user interface and raises the custom SelectionMade event.

> Like other client-side frameworks, Blazor has its own definition of custom components to speed up and simplify development.

Connecting to the Typeahead Component

A Blazor view that uses the Typeahead component will bind portions of its user interface to the SelectionMade event. Again, for the changes to take effect the StateHasChanged method should be invoked, with this code:

```
void ShowSelection(TypeAheadItem item)
{
  _countryName = item.DisplayText;
  _countryDescription = item.MenuText;
  this.StateHasChanged();
}
```

In the code snippet, data coming with the event is bound to the local properties of the view and once the DOM is refreshed the view is automatically updated (see **Figure 5**).

Wrapping Up

Modern Web front ends are more and more made of components. Components raise the abstraction level of the markup language and provide a much cleaner way to create Web content. Like other client-side frameworks, Blazor has its own definition of custom components to speed up and simplify development. The source code for this article can be found at bit.ly/2v0lYYZ, side by side with last month's column about using the JavaScript typeahead plug-in. ■

Dino Esposito *has authored more than 20 books and 1,000-plus articles in his 25-year career. Author of "The Sabbatical Break," a theatrical-style show, Esposito is busy writing software for a greener world as the digital strategist at BaxEnergy. Follow him on Twitter: @despos.*

Thanks to the following Microsoft technical expert for reviewing this article: *Daniel Roth*

JAMES MCCAFFREY

Autoencoders for Visualization Using CNTK

Suppose you have data that describes a set of people. Each person is represented by an age value and a height value. If you wanted to graph your data you could put age on the x-axis, height on the y-axis and use colored dots (say, blue for male and pink for female). The data has just two dimensions so there's no problem.

But what if your data has six dimensions, such as age, height, weight, income, debt, ZIP code? To graph such data, one approach is to condense the six-dimensional data down to two dimensions using a neural network autoencoder. You'll lose some information, but you'll be able to construct a 2D graph.

The best way to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo data has 1,797 items. Each item has 64 dimensions and falls into one of 10 classes. The demo creates a Microsoft Cognitive Toolkit (CNTK) neural network autoencoder to condense each item down to two dimensions, labeled as component1 and componet2, and then graphs the result. Interesting patterns in the data emerge. For example, data items that are class 0 (black dots) are quite different from class 7 items (orange dots). But class 8 items (red dots) have quite a bit of similarity with class 5 items (light green dots).

This article assumes you have intermediate or better programming skill with a C-family language and a basic familiarity with machine learning, but doesn't assume you know anything about autoencoders. All of the demo code is presented in this article. The complete source code and the data file used by the demo program are also available in the download that accompanies this article. All normal error checking has been removed to keep the main ideas as clear as possible.

Understanding the Data

The demo data looks like:

```
0,0,0,1,11,0,0,0,0,0,0,7,8, ... 16,4,0,0,4
0,0,9,14,8,1,0,0,0,0,12,14, ... 5,11,1,0,8
...
```

There are 1,797 data items, one per line, and each line has 65 comma-delimited values. Each line of data represents an 8x8 crude handwritten digit. The first 64 values on a line are grayscale pixel values between 0 and 16. The last value on a line is the digit value, so the first item has pixel values representing a "4" and the second item has pixel values for an "8."

The goal of the demo autoencoder is to reduce the 64 dimensions of a data item down to just two values so the item can be plotted as a point on an x-y graph. Even though the demo data represents an image, autoencoders can work with any kind of high-dimensionality numeric data.

The demo data is called the UCI digits dataset and can be found at bit.ly/2xDW3IY, or in the file download that accompanies this article.

Code download available at msdn.com/magazine/1218magcode.

Installing CNTK

Microsoft CNTK is a powerful open source neural network code library. CNTK is written in C++ for performance, but has a Python API for convenience and sanity. You don't install CNTK as a standalone system. First you install a distribution of Python that contains the core Python interpreter and several hundred additional packages. Then you install CNTK as a Python add-on package.

Installing CNTK can be tricky. For the demo, I first installed the Anaconda3 4.1.1 distribution for Windows, using the nice self-extracting executable at repo.continuum.io/archive. The distribution contains Python 3.5.2 and several packages required by CNTK.

Next, I installed CNTK v2.4 by downloading the compatible CPU-only (my machine doesn't have a GPU) CNTK .whl file to my local machine from bit.ly/2OfCCQg. Then I opened a command shell, navigated to the directory containing the CNTK .whl file and entered the command:

```
> pip install cntk-2.4-cp35-cp35m-win_amd64.whl
```

If you're new to Python, you can loosely think of a .whl file as somewhat similar to an .msi installation file and pip as somewhat similar to the Windows Installer program. Python is quite brittle with regard to versioning, so you have to be very careful to install compatible versions of Anaconda Python and CNTK. The vast majority of CNTK installation problems I see are directly related to version incompatibilities.

The Demo Program

The structure of the demo program, with a few minor edits to save space, is presented in the listing in **Figure 2**. I indent with two spaces rather than the usual four spaces to save space. And note that Python uses the "\" character for line continuation. I used Notepad to edit my program. Most of my colleagues prefer a more sophisticated editor, but I appreciate the simplicity of Notepad.

The demo program is named digits_autoenc.py and it starts by importing the NumPy, CNTK and Pyplot packages. NumPy enables basic numeric operations in Python and Pyplot is used to display the scatter plot shown in **Figure 1**. After a startup message, program execution begins by setting the global NumPy random seed so results will be reproducible.

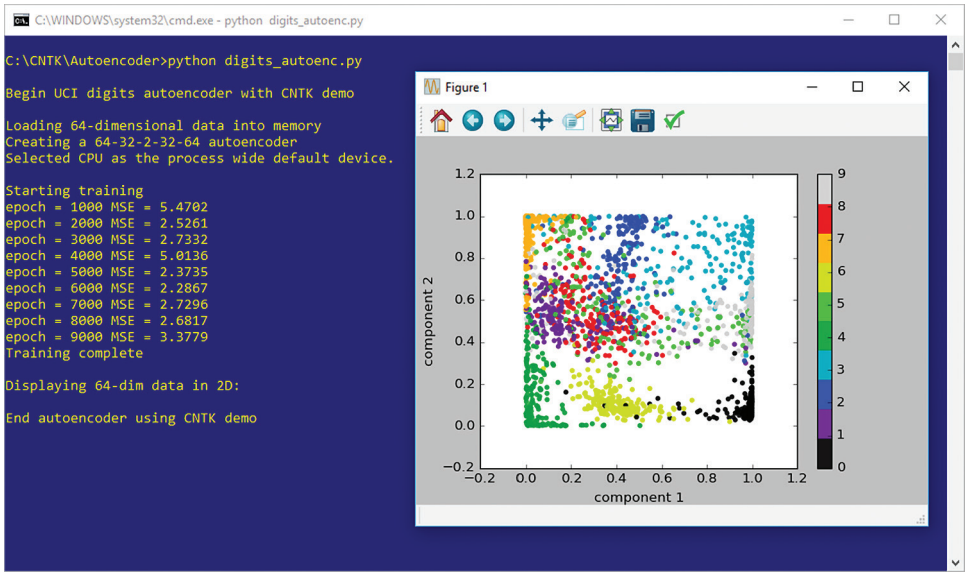


Figure 1 **Autoencoder Visualization Using CNTK Demo Run**

The demo loads the training data in memory using the NumPy loadtxt function:

```
data_file = ".\\Data\\digits_uci_test_1797.txt"
data_x = np.loadtxt(data_file, delimiter=",",
  usecols=range(0,64), dtype=np.float32)
labels = np.loadtxt(data_file, delimiter=",",
  usecols=[64], dtype=np.float32)
data_x = data_x / 16
```

The code assumes that the data is located in a subdirectory named Data. The loadtxt function has a lot of optional parameters. In this case, the function call specifies that the data is comma-delimited. The float32 datatype is the default for CNTK, so I could've omitted specifying it explicitly. The data_x object holds all 1,797 rows of the 64-pixel values and the labels object holds the corresponding 10 class values. The data_x object is modified by dividing by 16 so that all pixel values are scaled between 0.0 and 1.0.

Defining the Autoencoder

The demo creates a 64-32-2-32-64 neural network autoencoder model with these statements:

```
my_init = C.initializer.glorot_uniform(seed=1)
X = C.ops.input_variable(64, np.float32)  # inputs
layer1 = C.layers.Dense(32, init=my_init,
  activation=C.ops.sigmoid)(X)
layer2 = C.layers.Dense(2, init=my_init,
  activation=C.ops.sigmoid)(layer1)
layer3 = C.layers.Dense(32, init=my_init,
  activation=C.ops.sigmoid)(layer2)
layer4 = C.layers.Dense(64, init=my_init,
  activation=C.ops.sigmoid)(layer3)
```

The initializer object specifies the Glorot algorithm, which often works better than a uniform distribution on deep neural networks. The X object is set up to hold 64 input values. The next four layers create an autoencoder that crunches the 64 input values down to 32 values, and then crunches those down to just two values. An autoencoder is a specialized form of an encoder-decoder network. The diagram in **Figure 3** illustrates autoencoder architecture.

To keep the size of the diagram small, **Figure 3** shows a 6-3-2-3-6 autoencoder rather than the 64-32-2-32-64 architecture of the demo autoencoder.

Notice that the output values are the same as the input values so the autoencoder learns to predict its own inputs. The decoder part of the autoencoder expands the two values in the middle layer back up to the original 64 values. The net result of all this is that each 64-dimensional data item is mapped down to just two values. To get at these values the demo program defines:

```
enc_dec = C.ops.alias(layer4)
encoder = C.ops.alias(layer2)
```

In words, the encoder-decoder accepts values in the X object and generates output in layer4 with 64 nodes. The encoder accepts values in the X object and generates output in layer2 with two nodes.

The demo uses sigmoid activation on all layers. This results in all inner-layer nodes being between 0.0 and 1.0. There are many design options for autoencoders. You can use different numbers of inner layers and different numbers of nodes in each layer. You can use a different activation function, including the rarely used linear function. When using an autoencoder for dimensionality reduction for visualization, the quality of the resulting visualization is subjective, so your design choices are mostly a matter of trial and error.

Training the Autoencoder

Training is prepared using these seven statements:

```
Y = C.ops.input_variable(64, np.float32)  # targets
loss = C.squared_error(enc_dec, Y)
learner = C.adam(enc_dec.parameters, lr=0.005, momentum=0.90)
trainer = C.Trainer(enc_dec, (loss), [learner])
N = len(data_x)
bat_size = 2
max_epochs = 10000
```

Figure 2 **The Autoencoder Demo Program Structure**

```
# digits_autoenc.py
# condense the UCI digits to two dimensions
# CNTK 2.4 Anaconda3 4.1.1 (Python 3.5.2)

import numpy as np
import cntk as C
import matplotlib.pyplot as plt

def main():
  # 0. get started
  print("Begin UCI digits autoencoder with CNTK demo ")
  np.random.seed(1)

  # 1. load data into memory
  # 2. define autoencoder
  # 3. train model
  # 4. generate (x,y) pairs for each data item
  # 5. graph the data in 2D

  print("End autoencoder using CNTK demo ")

if __name__ == "__main__":
  main()
```

The Y object holds the same values as the X object and the training-loss function compares those values. The demo specifies squared error for the loss function, but because each value is between 0.0 and 1.0 due to the use of the sigmoid activation function, cross-entropy error could've been used.

For deep neural networks, the Adam (adaptive moment estimation) algorithm often performs better than basic stochastic gradient descent. The learning rate value (0.005) and momentum value (0.90) are hyperparameters and must be determined by trial and error. The batch size (two) and maximum number of training iterations (10,000) are also hyperparameters.

Training is performed like so:

```
for i in range(0, max_epochs):
  rows = np.random.choice(N, bat_size, replace=False)
  trainer.train_minibatch({ X: data_x[rows], Y: data_x[rows] })
  if i > 0 and i % int(max_epochs/10) == 0:
    mse = trainer.previous_minibatch_loss_average
    print("epoch = " + str(i) + " MSE = %0.4f " % mse)
```

On each training iteration, the random.choice function selects two rows from the 1,797 rows of data. This is a fairly crude approach because some rows may be selected more often than other rows. You could write a more sophisticated batching system, but for autoencoders the approach used by the demo is simple and effective.

The demo monitors training by displaying the squared error loss on the current batch of two items, every 10,000 / 10 = 1,000 epochs. The idea is to make sure that error tends to decrease during training; however, because the batch size is so small, there's quite a bit of fluctuation of the loss during training.

Using the Encoder

After training, the demo program generates (x, y) pairs for each of the 1,797 data items:

```
reduced = encoder.eval(data_x)
```

The return value is a matrix with 1,797 rows and two columns. Each row represents a reduced dimensionality version of the original data. The values in both columns are between 0.0 and 1.0 because the autoencoder uses sigmoid activation on all layers.

The demo program prepares the visualization with these statements:

```
print("Displaying 64-dim data in 2D: \n")
plt.scatter(x=reduced[:, 0], y=reduced[:, 1],
  c=labels, edgecolors='none', alpha=0.9,
  cmap=plt.cm.get_cmap('nipy_spectral', 10), s=20)
```

The x and y parameters of the scatter function are values for the x-axis and y-axis. The syntax reduced[:, 0] means all rows of the matrix but just the first column. The c parameter specifies colors. In this case the labels object, holding all of the 0 through 9 values for each data item, is passed to c.

The alpha parameter specifies the transparency level of each marker dot. The cmap parameter accepts a color mapping. The values 'nipy_spectral' and 10 mean to fetch 10 values from a spectral color gradient that ranges from 0 = black, through 4 = green, to 9 = gray. The Pyplot library supports many different color maps. The s parameter is the size of the marker dots, measured in pixels.

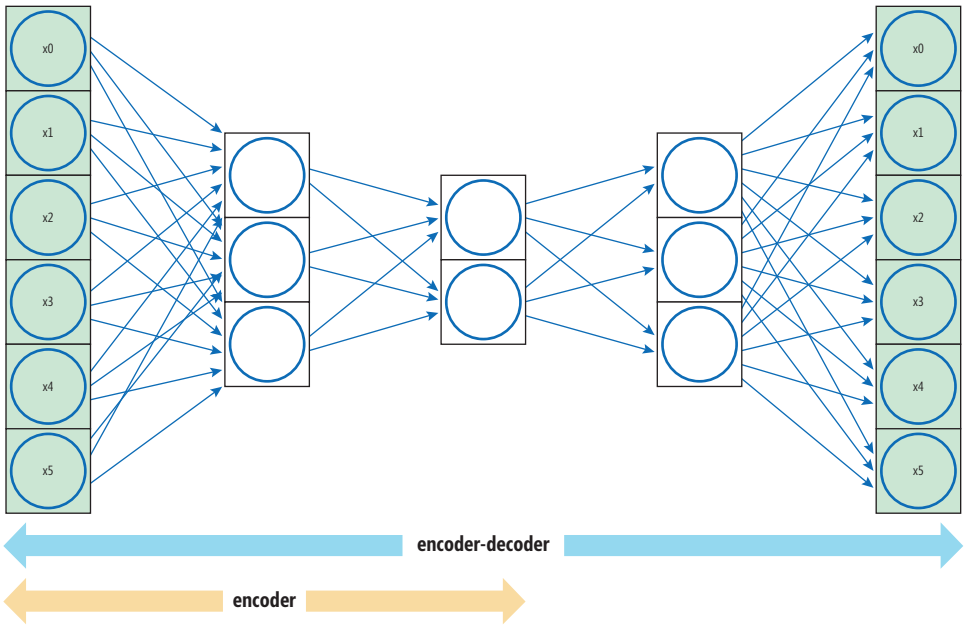


Figure 3 **Autoencoder Architecture**

After setting up the scatter plot, the demo program concludes like so:

```
  plt.xlabel('component 1')
  plt.ylabel('component 2')
  plt.colorbar()
  print("End autoencoder using CNTK demo ")
  plt.show()
if __name__ == "__main__":
  main()
```

The built-in colorbar function uses the 10 values from the nipy_spectral gradient. An alternative approach is to use the legend function.

Wrapping Up

There are several alternative approaches to dimensionality reduction for data visualization. Principal component analysis (PCA) is a classical statistics technique that has been used for decades. A newer technique, dating from 2008, is called t-distributed stochastic neighbor embedding (t-SNE) and it often works well, but only with relatively small datasets.

Autoencoders can be used for purposes other than dimensionality reduction for data visualization. For example, you could use an autoencoder to remove noise from images or documents. The idea is to reduce data down to some sort of core components, removing noise in the process, and then expand the data back, resulting in cleaner data in some sense. ■

Dr. James McCaffrey *works for Microsoft Research in Redmond, Wash. He has worked on several key Microsoft products, including Internet Explorer and Bing. Dr. McCaffrey can be reached at jamccaff@microsoft.com.*

Thanks to the following Microsoft experts who reviewed this article: *Chris Lee, Ricky Loynd*

#VSLive

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

April 22-26, 2019 | Hyatt Regency New Orleans

Spice Up Your Coding Skills in the Bayou

New Orleans

Intense Developer Training Conference

In-depth Technical Content On:

- AI, Data and Machine Learning
- Cloud, Containers and Microservices
- Delivery and Deployment
- Developing New Experiences
- DevOps in the Spotlight
- Full Stack Web Development
- .NET Core and More

New This Year!

On-Demand Session Recordings Now Available

Get on-demand access for one full year to all keynotes and sessions from Visual Studio Live! New Orleans, including everything Tuesday – Thursday at the conference.

Register by February 22 & Save Up To $400!

YOUR ADVENTURE STARTS HERE

vslive.com/neworleans

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360 | 1105 MEDIA

DAVID S. PLATT

Senioritis

It's hard being a senior citizen these days. Modern technology should greatly assist the elderly in their daily activities. But instead it is complex to the point of being counterproductive, increasing burdens rather than decreasing them. Let me explain.

My parents, both aged 84, live in a retirement community near me. They are some of the first digital immigrants. They will always speak geek with a heavy accent and a limited vocabulary. And they depend on their digital-native grandchildren to navigate anything complicated that they can't avoid. (See my June 2015 column at msdn.com/magazine/mt147245.)

They are a preview of the baby boomer bulge now peristalsing its way through the demographic snake. Approximately 15 percent of the U.S. population (46 million people) is over age 65 today. That's expected to rise to 24 percent (98 million) by 2060. This isn't an obscure edge case.

They describe it as a "simplified and easy experience for first-time Smartphone users," implying that it's something users should outgrow. Is that any way to treat a quarter of the U.S. population?

My father has trouble hearing, as do most people his age. Sometimes (with drama shows) he likes his TV to display closed captions, other times (football games) he doesn't. So he has to turn the captions on and off and on again. He has to find and follow the complex instructions every time, using the remote control with tiny buttons he can't manipulate, labeled with small text he can't read, working his way down through several on-screen menus—all without making a slip that would break something else, like resetting the TV's color temperature. He finds it very frustrating that this helpful feature exists, but is so difficult to access.

My parents have the same problem with smartphones. They bought iPhones because they like Apple's computers, but they can't read a regular iPhone screen. They can tap an icon, if they can recognize what it is without the tiny, illegible caption, but you can forget about typing any text in. Samsung's Galaxy line has an Easy Mode with larger icons and simplified choices. They describe it as a "simplified and easy experience for first-time Smartphone users," implying that it's something users should outgrow. Is that any way to treat a quarter of the U.S. population?

Voice control was supposed to solve all these problems, but, as always, it introduces others, no matter which vendor's wizard you're using. It's opaque, providing no cues to its capabilities, the way visual apps provide with controls. It times out quickly, which is especially frustrating for seniors who are puzzling about what to say. And it's brittle, requiring rigid speech patterns and timing, all difficult for seniors to deduce and repeat. Saying "Weather" often works. Saying "Text Bob to meet at 12 instead of 1, and nyah, nyah, nyah, your team lost," no chance. After the first few failures, they throw up their hands and say, to hell with it. And of course, anything audible is inherently difficult for hearing impaired people to use.

I'm convinced that only products designed specifically for seniors can solve their problems. I can only find one company that caters today's tech to seniors. Great Call (greatcall.com) makes a smartphone called the Jitterbug—the name comes from a 1934 song, signaling their target market. It offers the larger text and icons and touch zones that seniors need, simplifying access to camera, phone and mail.

There's a lot of money here if tech companies would just get on it. The Jitterbug is a fabulous marketing channel for value-added services tailored to seniors. For example, Great Call customers can get Urgent Care, which connects to a live nurse 24/7, for only five bucks more per month. I can imagine an add-on tech support service entitled Grandchild On Call.

The only thing preventing these seniors from rising up and hanging the perpetrators of bad technology is that they remember how things were before. "We didn't have TV when I was growing up, we had to fight over the family radio. And when we finally got a TV, it only received three channels, all black-and-white. And the family phone had a cord and a dial. Things could be a whole lot worse."

Best Buy recently purchased Great Call, suggesting a push into this underserved market. Combine that with tech support from its Geek Squad, and the company might be onto something. What other companies will be joining them? ■

DAVID S. PLATT *teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

#VSLIVE

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

MARCH 3-8, 2019 ➤ BALLY'S HOTEL AND CASINO

AN OASIS OF EDUCATION DAZZLING IN THE DESERT

Las Vegas

INTENSE DEVELOPER TRAINING CONFERENCE

In-depth Technical Content On:

- AI, Data and Machine Learning
- Cloud, Containers and Microservices
- Delivery and Deployment
- DevOps in the Spotlight
- Full Stack Web Development
- .NET Core and More
- Developing New Experiences

New This Year!

On-Demand Session Recordings Now Available

Get on-demand access for one full year to all keynotes and sessions from Visual Studio Live! Las Vegas, including everything Tuesday – Thursday at the conference.

Register by December 14 & Save Up To $500!

Your Adventure STARTS HERE!

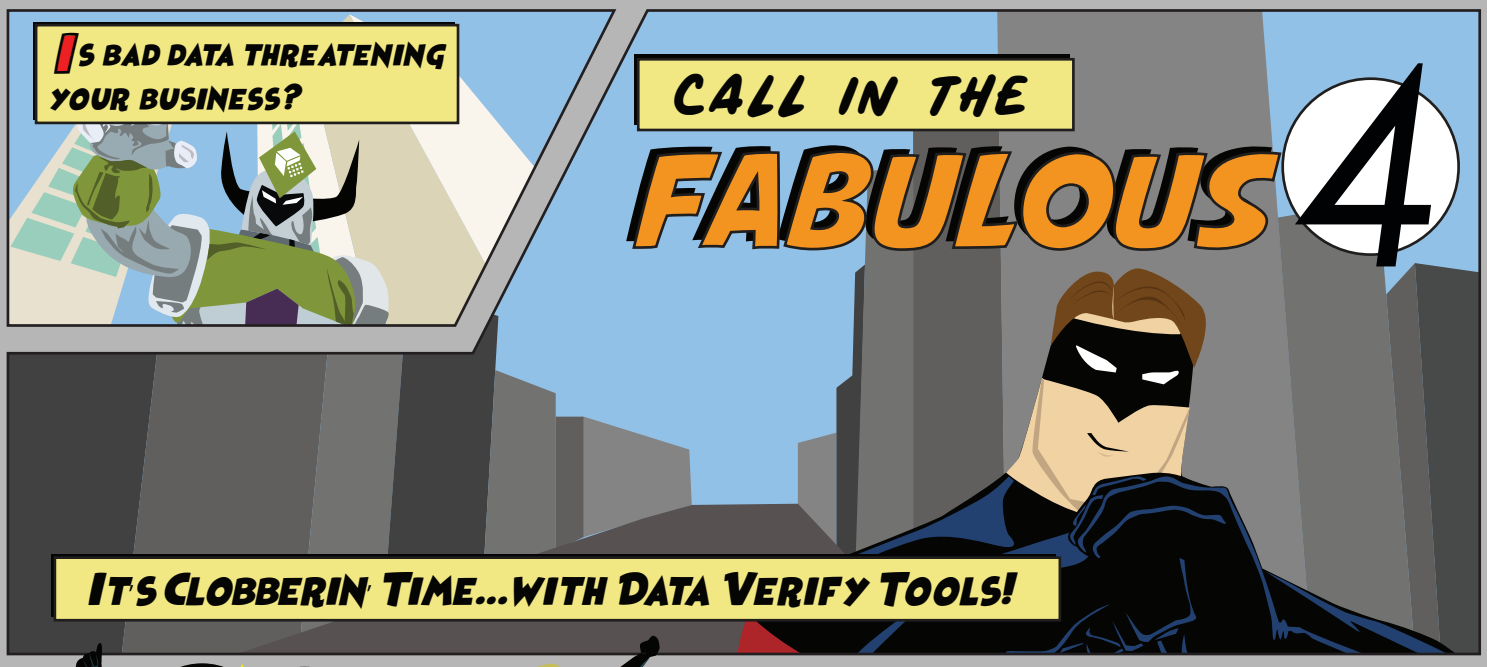
vslive.com/lasvegas

SUPPORTED BY

Microsoft Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360

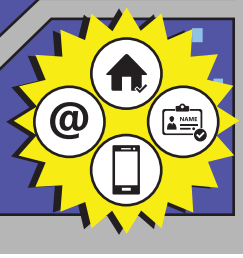


Visit Melissa Developer Portal to quickly combine our APIs (address, phone, email and name verification) and enhance ecommerce and mobile apps to prevent bad data from entering your systems.

With our toolsets, you'll be a data hero – preventing fraud, reducing costs, improving data for analytics, and increasing business efficiency.

- **Single Record & Batch Processing**
- **Scalable Pricing**
- **Flexible, Easy to Integrate Web APIs: REST, JSON & XML**
- **Other APIs available: Identity, IP, Property & Business**

Let's Team Up to Fight Bad Data Today!



melissadeveloper.com
1-800-MELISSA

melissa